File - /home/sebrimmer/Documents/Uni/Year_3/AE_402/HW/HW3/problem_1_lamberts.py

```python
"""
    File problem_1_lamberts.py created on 29/11/2021 by sebrimmer at 09:51:10

    Current Directory: HW3

    Use Lambert's problem to solve for the ΔV required to send a spacecraft from
    Earth to impact the asteroid Apophis. The spacecraft will depart Earth on
    May 1, 2024 and arrive at Apophis on June 15, 2025. The time-of-flight is 410 days.
    Use the JPL horizons website to query the position and velocity of Earth and Apophis
    on those days with respect to the solar system barycenter (SSB). When solving for the
    Lambert transfer consider only the gravity from the Sun and neglect the gravity of
    the Earth and Apophis. This type of mission is known as a kinetic impactor and is
    one approach for moving asteroids that are on a collision course with the Earth.

    Data from JPL Horizons Website in jpl_horizons_ssd_results.txt

"""
import numpy as np
from math import pi, sqrt, cos, sin, acos, asin, tan, atan


def sf_vector(vector_arr: np.ndarray, num_sig_fig) -> tuple:
    """
    Simple function that just returns vector array with values at a specfied number of sig figs
    :param vector_arr:      np vector array with float values to lots of sf
    :param num_sig_fig:     num sig fig to return values with
    :return:                tuple of prettied up nd.np array
    """
    # conver to list
    vector_arr = list(vector_arr)

    vector_arr = [round(val, num_sig_fig) for val in vector_arr]

    return tuple(vector_arr)


def main():

    au = 149_597_871  # km for 1 AU for canonical units
    du = 1 * au

    tu = sqrt(du**3 / 1.327e11)

    tof_days = 410  # desired tof as stated in question
    tof_seconds = tof_days * 24 * 60 * 60  # converting tof into seconds

    # Absolute position and velocity of Earth at t = 0
    r_earth_vector_t0_abs = np.array([-1.152298994309664E+08,
                                      -9.900155838813813E+07,
                                      3.696167672807723E+04])
    r_earth_vector_t0_canon = r_earth_vector_t0_abs /au

    v_earth_vector_t0_abs = np.array([1.897300201461335E+01,
                                      -2.268665080580648E+01,
                                      5.966729305662000E-04])
    v_earth_vector_t0_canon = v_earth_vector_t0_abs * tu/au

    # Absolute position of Apophis at t = 410 days (impact time)
    r_apophis_vector_t1_abs = np.array([-7.850925795703618E+07,
                                        1.374546686841051E+08,
                                        -9.195926177815042E+06])
    r_apophis_vector_t1_canon = r_apophis_vector_t1_abs /au

    r_earth_mag_t0_abs = np.linalg.norm(r_earth_vector_t0_abs)
    r_apophis_mag_t1_abs = np.linalg.norm(r_apophis_vector_t1_abs)

    # Converting position vectors into canonical units with AU
    r1 = np.linalg.norm(r_earth_vector_t0_canon)              # r1
    r2 = np.linalg.norm(r_apophis_vector_t1_canon)          # r2

    mu = 1

    # theta angle between the two radius vectors
    dot_product = np.dot(r_earth_vector_t0_abs, r_apophis_vector_t1_abs)
    theta = np.arccos(dot_product / (r_earth_mag_t0_abs * r_apophis_mag_t1_abs))
    theta = 2*pi - theta
```

```
77
78      # third side, c, of the space triangle
79      c = sqrt(r1**2 + r2**2 - 2 * r1 * r2 * cos(theta))
80      chord_unit_vector = (r_apophis_vector_t1_canon-r_earth_vector_t0_canon)/c
81
82      # space triangle semi-perimeter
83      s = (r1 + r2 + c) / 2
84
85      # compute minimum transfer time
86      tp = sqrt(2)/(3*sqrt(mu)) * (s**1.5 - np.sign(sin(theta)) * (s - c)**1.5)
87
88      if tp < tof_seconds/tu:
89          transfer_poss = True
90      else:
91          transfer_poss = False
92
93      # Minimum semi-major axis
94      a_m = s/2
95
96      # Initial values of alpha and beta based on am for t_min
97      a_0 = 2 * asin(sqrt(s / (2 * a_m)))
98      b_0 = - 2 * asin(sqrt((s-c)/(2 * a_m)))
99
100     # t_m corresponding to a_m is
101     t_m = sqrt(s**3/8) * ( pi - b_0 + sin(b_0))
102
103     # Now solve Lambert's equation for a. After iteration in matlab, a = 1.2478
104     a = 1.2378
105
106     # re-calculate a_0 and b_0 values based on new a in the equation
107     a_0 = 2 * asin(sqrt(s / (2 * a)))
108     b_0 = 2 * asin(sqrt((s-c)/(2 * a)))
109
110     # t_m of 158.3586 days means our transfer time of 410 days is on the upper branch
111     # theta > pi , so beta = - b_0, and t_f > t_m so alpha = 2pi - a_0
112     alpha = 2*pi - a_0
113     beta = - b_0
114
115     # Work out A and B constants for velocity vectors
116     A = sqrt(1/(4*a)) * 1/tan(alpha * 0.5)
117     B = sqrt(1/(4*a)) * 1/tan(beta * 0.5)
118
119     v1 = (B+A) * chord_unit_vector + (B-A) * r_earth_vector_t0_canon/r1
120
121     delta_v = v1 - v_earth_vector_t0_canon
122
123     # Output
124     out_string = f"Distance Unit 1-AU: {au:.0f} km\n" \
125                  f"Time Unit 1-TU:     {tu:.0f} seconds\n" \
126                  f"-------------------------------------------------\n" \
127                  f"Earth r1 at t0: {r1:.4f}\n" \
128                  f"Apophis r2 at t1: {r2:.4f}\n" \
129                  f"1.1) Magnitude of Earth's position vector (AU): {sf_vector(
    r_earth_vector_t0_canon/r1, 4)}\n" \
130                  f"1.2) Magnitude of Apophis' position vector (AU): {sf_vector(
    r_apophis_vector_t1_canon/r2, 4)}\n" \
131                  f"1.3) Chord (AU): {c:.4f}\n" \
132                  f"1.4) Semiperimeter (AU): {s:.4f}\n" \
133                  f"1.5) Minimum semimajor axis (AU): {a_m:.4f}\n" \
134                  f"1.6) Angle between the position vectors (radians). You may need to use 2*pi-
    theta to\n" \
135                  f"     prevent a retrograde orbit transfer. As a check, the z-component of the
    angular\n" \
136                  f"     momentum vector will be positive for prograde orbits and negative for
    retrograde\n" \
137                  f"     orbits.\n" \
138                  f"     Theta: {theta:.4f}\n" \
139                  f"1.7) Minimum transfer time: {t_m:.4f} time units ({t_m * tu / (60 * 60 * 24) :.
    4f} days)\n" \
140                  f"1.8) Semimajor axis (AU) after iterating (if using fzero in MATLAB, " \
141                  f"use a_guess = 1.1 AU) {a:.4f}\n" \
142                  f"1.9) Unit vector in the direction of r1 {sf_vector(v_earth_vector_t0_canon, 6)
    }\n" \
143                  f"1.10) Unit vector in the direction of r2 {sf_vector(v_earth_vector_t0_canon, 6)
    }\n" \
144                  f"1.11) u_c (see notes) {sf_vector(v_earth_vector_t0_canon, 6)}\n" \
```

```
145                 f"1.12) alpha (radians) {alpha:.6f}\n" \
146                 f"1.13) beta (radians) {beta:.6f}\n" \
147                 f"1.14) A (AU/TU) (see notes): {A:.8f}\n" \
148                 f"1.15) B (AU/TU) (see notes): {B:.8f}\n" \
149                 f"1.16) Velocity at the start of the Lambert transfer (AU/TU), after the burn: "
     \
150                 f"{sf_vector(v1, 4)}\n" \
151                 f"1.17) Earth's velocity at the time of departure (AU/TU): {sf_vector(
    v_earth_vector_t0_canon, 6)}\n" \
152                 f"1.18) Departure delta V (km/s) at departure: \n" \
153                 f"      Delta-V vector (AU/TU) : {sf_vector(delta_v, 4)}\n" \
154                 f"      Delta-V magnitude (AU/TU) : {np.linalg.norm(delta_v):.4f}\n" \
155                 f"      Delta-V magnitude absolute : {np.linalg.norm(delta_v * au/tu):.4f}\n" \
156                 f"\nTOF corresponding to a_m: {t_m:.4f} time units, {t_m * tu / (60 * 60 * 24) :.
    4f} days\n" \
157                 f"-----------------------------------------------------------------\n" \
158                 f"Quick h check for pro-grade; z of h should be positive. " \
159                 f"{sf_vector(np.cross(r_earth_vector_t0_canon, v1), 4)}\n" \
160                 f"-----------------------------------------------------------------\n" \
161
162
163     print(out_string)
164     with open('output/problem_1_output.txt', 'w') as output:
165         output.write(out_string)
166
167     return 0
168
169
170 if __name__ == '__main__':
171     main()
172
```

File - /home/sebrimmer/Documents/Uni/Year_3/AE_402/HW/HW3/problem_2_transfers.py

```python
"""
    File problem_2_transfers.py created on 29/11/2021 by sebrimmer at 15:04:39

    Current Directory: HW3

    Compute the total ΔV required to send a spacecraft a from a geocentric circular
    orbit of 7000 km radius to geocentric circular orbit of 105000 km radius using
    a Hohmann transfer. Repeat the ΔV computation using a bi-elliptic transfer where
    apogee of the intermediate orbit is 210000 km. Which approach requires less ΔV?
    At what value, to the nearest km, of apogee of the intermediate orbit does the
    other approach require the least ΔV?

"""
import numpy as np
from math import pi, sqrt, cos, sin, acos, asin, tan, atan


def main():

    mu_earth = 398_600
    r_1 = 7_000
    r_2 = 105_000

    bi_elip_intermed_apogee = 210_000

    # Hohmann transfer semi-major axis
    a_h = (r_1 + r_2) / 2

    # Assuming circular orbits, the velocities at periapse/apoapse are:
    v_periapse = sqrt(mu_earth * (2 / r_1 - 1 / a_h))
    v_apoapse = sqrt(mu_earth * (2 / r_2 - 1 / a_h))

    orbit_1_circ_velocity = sqrt(mu_earth / r_1)
    orbit_2_circ_velocity = sqrt(mu_earth / r_2)

    delta_v_a = v_periapse - orbit_1_circ_velocity
    delta_v_b = orbit_2_circ_velocity - v_apoapse
    delta_v_total_hohmann = delta_v_a + delta_v_b

    # Now performing calculations for bi-elliptic transfer
    a_h_be_first = (r_1 + bi_elip_intermed_apogee) / 2  # b-e semi-major axis for first leg
    a_h_be_apogee_first = a_h_be_first * 2 - r_1  # b-e apogee, calculated from semi-major axis and r1

    # Velocity at perigee on the first leg (i.e. first Hohmann transfer) for the Bielliptic transfer (km/s)
    v_be_periapse = sqrt(mu_earth * (2 / r_1 - 1 / a_h_be_first))

    # Velocity at apogee on the first leg (i.e. first Hohmann transfer) for the Bielliptic transfer (km/s)
    v_be_apoapse = sqrt(mu_earth * (2 / a_h_be_apogee_first - 1 / a_h_be_first))

    # Delta V to get from the circular 7000 km orbit onto the first leg (i.e. first Hohmann transfer)
    # for the Bielliptic transfer:
    # delta v = what we want - what we have
    delta_v_a_be_to_first_leg = v_be_periapse - orbit_1_circ_velocity

    # Semimajor axis for transfer from the 210000 km orbit to the 105000 km orbit (km)
    a_h_be_second = (a_h_be_apogee_first + r_2) / 2

    # Velocity at apogee on the second leg (i.e. second Hohmann transfer) for the Bielliptic transfer (km/s)
    v_be_apogee_second = sqrt(mu_earth * (2 / a_h_be_apogee_first - 1 / a_h_be_second))
    v_be_perigee_second = sqrt(mu_earth * ((2 / r_2) - (1 / a_h_be_second)))

    # Delta V to transfer from the first leg (i.e. first Hohmann transfer)
    # to the second leg (i.e. second Hohmann transfer) for the Bielliptic transfer (km/s)
    delta_v_b_be_to_second_leg = v_be_apogee_second - v_be_apoapse

    # Delta V to get from the second leg (i.e. second Hohmann transfer)
    # onto the circular orbit of radius 105000 km (km/s)
    delta_v_b_be_to_second_orbit = v_be_perigee_second - orbit_2_circ_velocity

    # Total DV for the Bielliptic transfer (km/s)
    # total dv = dv to get onto first transfer ellipse
```

```
72      #             + dv to get onto second transfer ellipse
73      #             + dv to get onto second orbit
74      delta_v_total_bi_elliptic = delta_v_a_be_to_first_leg \
75                                  + delta_v_b_be_to_second_leg \
76                                  + delta_v_b_be_to_second_orbit
77
78      # Radius of intermediate orbit such that the other transfer option can be calculated
    iteratively or analytically.
79      # We use the iterative method, and decrease the intermediate orbit semi-maj axis until the
    delta v values match
80
81      bi_elip_intermed_apogee = 210_000
82      delta_v_total_bi_elliptic = 4.029   # initial delta v for hohmann is 4.046
83
84      while delta_v_total_bi_elliptic <= delta_v_total_hohmann:
85
86          a_h_be_first_iter = (r_1 + bi_elip_intermed_apogee) / 2  # b-e semi-major axis for first
    leg
87          a_h_be_apogee_first_iter = a_h_be_first_iter * 2 - r_1  # b-e apogee, calculated from semi
    -major axis and r1
88
89          # Velocity at perigee on the first leg (i.e. first Hohmann transfer) for the Bielliptic
    transfer (km/s)
90          v_be_periapse_iter = sqrt(mu_earth * (2 / r_1 - 1 / a_h_be_first_iter))
91
92          # Velocity at apogee on the first leg (i.e. first Hohmann transfer) for the Bielliptic
    transfer (km/s)
93          v_be_apoapse_iter = sqrt(mu_earth * (2 / a_h_be_apogee_first_iter - 1 / a_h_be_first_iter
    ))
94
95          # Delta V to get from the circular 7000 km orbit onto the first leg (i.e. first Hohmann
    transfer)
96          # for the Bielliptic transfer:
97          # delta v = what we want - what we have
98          delta_v_a_be_to_first_leg_iter = v_be_periapse_iter - orbit_1_circ_velocity
99
100         # Semimajor axis for transfer from the 210000 km orbit to the 105000 km orbit (km)
101         a_h_be_second_iter = (a_h_be_apogee_first_iter + r_2) / 2
102
103         # Velocity at apogee on the second leg (i.e. second Hohmann transfer) for the Bielliptic
    transfer (km/s)
104         v_be_apogee_second_iter = sqrt(mu_earth * (2 / a_h_be_apogee_first_iter - 1 /
    a_h_be_second_iter))
105         v_be_perigee_second_iter = sqrt(mu_earth * ((2 / r_2) - (1 / a_h_be_second_iter)))
106
107         # Delta V to transfer from the first leg (i.e. first Hohmann transfer)
108         # to the second leg (i.e. second Hohmann transfer) for the Bielliptic transfer (km/s)
109         delta_v_b_be_to_second_leg_iter = v_be_apogee_second_iter - v_be_apoapse_iter
110
111         # Delta V to get from the second leg (i.e. second Hohmann transfer)
112         # onto the circular orbit of radius 105000 km (km/s)
113         delta_v_b_be_to_second_orbit_iter = v_be_perigee_second_iter - orbit_2_circ_velocity
114
115         # Total DV for the Bielliptic transfer (km/s)
116         # total dv = dv to get onto first transfer ellipse
117         #             + dv to get onto second transfer ellipse
118         #             + dv to get onto second orbit
119         delta_v_total_bi_elliptic = delta_v_a_be_to_first_leg_iter \
120                                     + delta_v_b_be_to_second_leg_iter \
121                                     + delta_v_b_be_to_second_orbit_iter
122
123         # decrement the bi_elip_intermed_apogee variable for the next iteration
124         bi_elip_intermed_apogee -= 1
125
126     output_string = f"Hohmann transfer semi-major axis: {a_h} km\n" \
127                     f"Apoapse Velocity: {v_apoapse:.3f} km\n" \
128                     f"Periapse Velocity: {v_periapse:.3f} km\n" \
129                     f"Orbit 1 circ velocity: {orbit_1_circ_velocity:.3f}\n" \
130                     f"Orbit 2 circ velocity: {orbit_2_circ_velocity:.3f}\n" \
131                     f"Delta v1 Velocity: {delta_v_a:.3f} km\n" \
132                     f"Delta v2 Velocity: {delta_v_b:.3f} km\n" \
133                     f"Total dV: {delta_v_total_hohmann:.3f} km\n\n" \
134                     f"Hohmann transfer semi-major axis (bi-elliptic): {a_h_be_first} km\n" \
135                     f"2.07) Semimajor axis for transfer from the 7000 km orbit to the " \
136                     f"210000 km orbit (km): {a_h_be_first:.3f}\n" \
137                     f"" \
```

```
138                     f"2.08) Velocity at perigee on the first leg (i.e. first Hohmann transfer)  \n
    " \
139                     f"      for the Bielliptic transfer (km/s): {v_be_periapse:.3f}\n" \
140                     f"2.09) Delta V to get from the circular 7000 km orbit onto the first leg\n" \
141                     f"      (i.e. first Hohmann transfer) for the Bielliptic transfer: " \
142                     f"{delta_v_a_be_to_first_leg:.3f} km/s\n\n" \
143                     f"2.10) Velocity at apogee on the first leg (i.e. first Hohmann transfer)\n" \
144                     f"      for the Bielliptic transfer (km/s): {v_be_apoapse:.3f} km/s\n\n" \
145                     f"2.11) Semimajor axis for transfer from the 210000 km orbit to the 105000 km
     " \
146                     f"orbit: {a_h_be_second} km/s\n\n" \
147                     f"2.12) Velocity at apogee on the second leg (i.e. second Hohmann transfer)\n"
     \
148                     f"      for the Bielliptic transfer (km/s): {v_be_apogee_second:.3f} km/s\n\n"
     \
149                     f"2.13) Delta V to transfer from the first leg (i.e. first Hohmann transfer)\n
    " \
150                     f"      to the second leg (i.e. second Hohmann transfer) for the Bielliptic \n
    " \
151                     f"      transfer: {delta_v_b_be_to_second_leg:.3f} km/s\n\n" \
152                     f"2.14) Velocity at perigee on the second leg (i.e. second Hohmann transfer)\n
    " \
153                     f"      for the Bielliptic transfer: {v_be_perigee_second:.3f} km/s\n\n" \
154                     f"2.15) Delta V to get from the second leg (i.e. second Hohmann transfer)\n" \
155                     f"      onto the circular orbit of radius 105000 km: {
    delta_v_b_be_to_second_orbit:.3f} km/s\n\n" \
156                     f"2.16) Total DV for the Bielliptic transfer: {delta_v_total_bi_elliptic:.3f}
    km/s\n\n" \
157                     f"2.17) Radius of intermediate orbit such that the other transfer option
    requires " \
158                     f"less DV: {bi_elip_intermed_apogee:.3f} km " \
159
160     print(output_string)
161
162     with open('output/problem_2_output.txt', 'w') as output:
163         output.write(output_string)
164
165     return 0
166
167
168 if __name__ == '__main__':
169     main()
170
```

File - /home/sebrimmer/Documents/Uni/Year_3/AE_402/HW/HW3/matlab_iterator_for_a.m

```
% Lambert Solver for HW3 Problem_1 - Earth-Apophis Intercept

clear all;
close all; clc;

a = fzero(@lambert, 1.1)

function f = lambert(a)

s  = 1.8381;        % Pre-calculated space-triangle semi-perimeter
c  = 1.6007;        % Pre-calculated chord length
tf = (410/365.25) * 2 * pi;     % ToF of 410 days

% Updated value for alpha because our t_f is greater than t_m
alpha = 2*pi - 2*asin(sqrt(s/(2*a)));
beta  = - 2*asin(sqrt((s-c)/(2*a)));

f = tf-(a^(3/2))*(alpha-beta-sin(alpha)+sin(beta));

end

```