```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 □□ 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
```

```python
    :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

    # third side, c, of the space triangle
    c = sqrt(r1**2 + r2**2 - 2 * r1 * r2 * cos(theta))
    chord_unit_vector = (r_apophis_vector_t1_canon-r_earth_vector_t0_canon)/c
```

```python

    # space triangle semi-perimeter
    s = (r1 + r2 + c) / 2

    # compute minimum transfer time
    tp = sqrt(2)/(3*sqrt(mu)) * (s**1.5 - np.sign(sin(theta)) * (s - c)**1.5)

    if tp < tof_seconds/tu:
        transfer_poss = True
    else:
        transfer_poss = False

    # Minimum semi-major axis
    a_m = s/2

    # Initial values of alpha and beta based on am for t_min
    a_0 = 2 * asin(sqrt(s / (2 * a_m)))
    b_0 = - 2 * asin(sqrt((s-c)/(2 * a_m)))

    # t_m corresponding to a_m is
    t_m = sqrt(s**3/8) * ( pi - b_0 + sin(b_0))

    # Now solve Lambert's equation for a. After iteration in matlab, a = 1.2478
    a = 1.2378

    # re-calculate a_0 and b_0 values based on new a in the equation
    a_0 = 2 * asin(sqrt(s / (2 * a)))
    b_0 = 2 * asin(sqrt((s-c)/(2 * a)))

    # t_m of 158.3586 days means our transfer time of 410 days is on the upper branch
    # theta > pi , so beta = - b_0, and t_f > t_m so alpha = 2pi - a_0
    alpha = 2*pi - a_0
    beta = - b_0

    # Work out A and B constants for velocity
```

```
vectors
    A = sqrt(1/(4*a)) * 1/tan(alpha * 0.5)
    B = sqrt(1/(4*a)) * 1/tan(beta * 0.5)

    v1 = (B+A) * chord_unit_vector + (B-A) * r_earth_vector_t0_canon/r1

    delta_v = v1 - v_earth_vector_t0_canon

    # Output
    out_string = f"Distance Unit 1-AU: {au:.0f} km\n" \
                 f"Time Unit 1-TU:     {tu:.0f} seconds\n" \
                 f"---------------------------------------------------\n" \
                 f"Earth r1 at t0: {r1:.4f}\n" \
                 f"Apophis r2 at t1: {r2:.4f}\n" \
                 f"1.1) Magnitude of Earth's position vector (AU): {sf_vector(r_earth_vector_t0_canon/r1, 4)}\n" \
                 f"1.2) Magnitude of Apophis' position vector (AU): {sf_vector(r_apophis_vector_t1_canon/r2, 4)}\n" \
                 f"1.3) Chord (AU): {c:.4f}\n" \
                 f"1.4) Semiperimeter (AU): {s:.4f}\n" \
                 f"1.5) Minimum semimajor axis (AU): {a_m:.4f}\n" \
                 f"1.6) Angle between the position vectors (radians). You may need to use 2*pi-theta to\n" \
                 f"     prevent a retrograde orbit transfer. As a check, the z-component of the angular\n" \
                 f"     momentum vector will be positive for prograde orbits and negative for retrograde\n" \
                 f"     orbits.\n" \
                 f"     Theta: {theta:.4f}\n" \
```

```
                 f"1.7) Minimum transfer time: {t_m:.4f} time units ({t_m * tu / (60 * 60 * 24) :.4f} days)\n" \
                 f"1.8) Semimajor axis (AU) after iterating (if using fzero in MATLAB, use a_guess = 1.1 AU) {a:.4f}\n" \
                 f"1.9) Unit vector in the direction of r1 {sf_vector(v_earth_vector_t0_canon, 6)}\n" \
                 f"1.10) Unit vector in the direction of r2 {sf_vector(v_earth_vector_t0_canon, 6)}\n" \
                 f"1.11) u_c (see notes) {sf_vector(v_earth_vector_t0_canon, 6)}\n" \
                 f"1.12) alpha (radians) {alpha:.6f}\n" \
                 f"1.13) beta (radians) {beta:.6f}\n" \
                 f"1.14) A (AU/TU) (see notes): {A:.8f}\n" \
                 f"1.15) B (AU/TU) (see notes): {B:.8f}\n" \
                 f"1.16) Velocity at the start of the Lambert transfer (AU/TU), after the burn: {sf_vector(v1, 4)}\n" \
                 f"1.17) Earth's velocity at the time of departure (AU/TU): {sf_vector(v_earth_vector_t0_canon, 6)}\n" \
                 f"1.18) Departure delta V (km/s) at departure: \n" \
                 f"      Delta-V vector (AU/TU) : {sf_vector(delta_v, 4)}\n" \
                 f"      Delta-V magnitude (AU/TU) : {np.linalg.norm(delta_v):.4f}\n" \
                 f"      Delta-V magnitude absolute : {np.linalg.norm(delta_v * au/tu):.4f}\n" \
                 f"\nTOF corresponding to a_m: {t_m:.4f} time units, {t_m * tu / (60 * 60 * 24) :.4f} days\n" \
                 f
"---------------------------------------------------
```

```
---------------\n" \
                 f"Quick h check for pro-grade; z of h should be positive. {sf_vector(np.cross(r_earth_vector_t0_canon, v1), 4)}\n" \
                 f"------------------------------------------------------------------\n" \


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

    return 0


if __name__ == '__main__':
    main()

```