MAE 146. Astronautics: Project

Winter Quarter 2026

Instructions

- There are 4 problems in the project.
- In problems 1, 2, & 3, you will be asked to implement fundamental Astrodynamics programs in restricted two-body dynamics.
- In problem 4, you will be using functions from problems 1 through 3 to perform preliminary mission analysis to a Near-Earth object (NEO).
- Submit the following:
  1. a zip folder that includes all your code files for functions from problems 1-3 and script(s) for problem 4, and
  2. a PDF report summarizing your answers to problem 4.
- The project is graded in terms of code quality, automated tests, and your report for problem 4. For code quality, we will check that (i) your functions abide by the signatures from each question, and (ii) your functions and scripts are cleanly written & commented appropriately.

Implementation quality (8 functions & script for problem 4)	/10
Automated tests	/20
Report for problem 4	/20
Total	/50

Problem 1: Keplerian Elements

Implement functions to

(a) Convert a state vector $\boldsymbol{x} = [\boldsymbol{r}; \boldsymbol{v}] \in \mathbb{R}^6$ in the inertial frame to Keplerian elements $\{a, e, i, \Omega, \omega, \theta\}$. The function should have the following signature:

```
function kep = rv2kep(rv, MU)
    arguments (Input)
        rv (6,1) double     % cartesian vector
        MU (1,1) double     % gravitational parameter
    end

    arguments (Output)
        kep (6,1) double    % Keplerian elements vector [a; e; i; W; w; ta]
    end

    % (fill in your code)
    % ...
end
```

(b) Convert Keplerian elements $\{a, e, i, \Omega, \omega, \theta\}$ to the corresponding inertial state vector $\boldsymbol{x} = [\boldsymbol{r}; \boldsymbol{v}] \in \mathbb{R}^6$. The function should have the following signature:

```
function rv = kep2rv(kep, MU)
    arguments (Input)
        kep (6,1) double     % Keplerian elements vector
        MU  (1,1) double     % gravitational parameter
    end

    arguments (Output)
        rv (6,1) double    % return 6-by-1 state vector
    end

    % (fill in your code)
    % ...
end
```

Problem 2: Kepler's Problem

Implement functions to

(a) Convert true anomaly θ to mean anomaly M. The function should have the following signature:

```
function ma = ta2ma(e, ta)
    arguments (Input)
        e  (1,1) double     % eccentricity
        ta (1,1) double     % true anomaly
    end

    arguments (Output)
        ma (1,1) double    % mean anomaly
    end

    % (fill in your code)
    % ...
end
```

(b) Convert mean anomaly M to true anomaly θ. The function should have the following signature:

```
function ta = ma2ta(e, ma)
    arguments (Input)
        e  (1,1) double     % eccentricity
        ma (1,1) double     % mean anomaly
    end

    arguments (Output)
        ta (1,1) double    % true anomaly
    end

    % (fill in your code)
    % ...
end
```

(c) Compute the true anomaly after time of flight `tof` along a circular/elliptical orbit. The function should have the following signature:

```
function ta = propagate_elliptical_ta(period, e, ta0, tof)
    arguments (Input)
        period (1,1) double    % period
        e      (1,1) double    % eccentricity
        ta0    (1,1) double    % initial true anomaly
        tof    (1,1) double    % time of flight
    end

    arguments (Output)
        ta (1,1) double        % true anomaly after tof
    end

    assert(e < 1)

    % (fill in your code)
    % ...
end
```

(d) Propagate the Cartesian state $\boldsymbol{x}(t_0) = [\boldsymbol{r}(t_0); \boldsymbol{v}(t_0)]$ over time `tof` and return the final state $\boldsymbol{x}(\texttt{tof}) = [\boldsymbol{r}(\texttt{tof}); \boldsymbol{v}(\texttt{tof})]$. Within the function, you should make use of the functions you've previously employed. The function should have the following signature:

```
function rvf = propagate_elliptical(rv, tof, MU)
    arguments (Input)
        rv  (6,1) double     % initial cartesian vector
        tof (1,1) double     % time of flight
```

```
        MU  (1,1) double     % gravitational parameter
    end

    arguments (Output)
        rvf (6,1) double        % final position & velocity vector [rf; vf]
    end

    assert(e < 1)

    % (fill in your code)
    % ...
end
```

Problem 3: Gauss' Problem

Implement functions to

1. Evaluate the Stumpff functions $C(z)$ and $S(z)$. The function should have the following signature:

```
function [C,S] = stumpff(z,eps)
    arguments (Input)
        z   (1,1) double    % function input
        eps (1,1) double    % tolerance for Stumpff functions near 0
    end

    arguments (Output)
        C (1,1) double      % Stumpff function C evaluated at z
        S (1,1) double      % Stumpff function S evaluated at z
    end

    % (fill in your code)
    % ...
end
```

2. Solve Lambert's problem using universal variables. The function should have the following signature:

```
function [v1,v2,residual] = lambert_universal(r1,r2,tof,MU,cw,eps,tol)
    arguments (Input)
        r1  (3,1) double    % initial position vector
        r2  (3,1) double    % final position vector
        tof (1,1) double    % time of flight
        MU  (1,1) double    % gravitational parameter
        cw  (1,1) double    % whether to use clockwise path, 0 or 1
        eps (1,1) double    % tolerance for Stumpff functions near 0
        tol (1,1) double    % tolerance on root-solving
    end

    arguments (Output)
        v1 (3,1) double         % initial velocity vector
        v2 (3,1) double         % final velocity vector
        residual (1,1) double   % residual on converged root-solving problem
    end

    % (fill in your code)
    % ...
end
```

Problem 4: Mission to Didymos (20 pt)

We will now perform preliminary mission analysis for a mission to the Near-Earth object (NEO) 65803 Didymos. The orbital elements of Earth and Didymos at some reference time, t_{ref}, are

$$\{a, e, i, \Omega, \omega, \theta(t_{\text{ref}})\}_{\text{Earth}} = \{1.00000011\,\text{AU},\ 0.016710212,\ 0.00005^\circ,\ -11.26064^\circ,\ 102.94719^\circ,\ 100.46435^\circ\}$$
$$\{a, e, i, \Omega, \omega, \theta(t_{\text{ref}})\}_{\text{Didymos}} = \{1.6442\,\text{AU}, 0.38385,\ 3.4079^\circ,\ 73.196^\circ,\ 319.321^\circ,\ 195.0^\circ\}$$

where $1\,\text{AU} = 149.6 \times 10^6$ km, $\mu = 132712000000$ km^3/s^2, and $1\,\text{day} = 86400\,\text{sec}$.

(a) (3 pt) What are the position and velocity vectors of Earth on $t = t_{\text{ref}} + 12$ days?

(b) (2 pt) What are the position and velocity vectors of Didymos on $t = t_{\text{ref}} + 412$ days?

(c) (5 pt) Consider transferring from Earth to Didymos, with a launch date $t_0 = t_{\text{ref}} + 12$ days and arrival date $t_f = t_{\text{ref}} + 412$ days. Plot the transfer along with the orbit of Earth and Didymos in 3D, and calculate the total transfer ΔV magnitude in km/s. You may assume a zero-sphere of influence at both Earth and Didymos, i.e., the total ΔV is

$$\Delta V = \Delta V_{\text{Earth}} + \Delta V_{\text{Didymos}} = \|\boldsymbol{v}_1 - \boldsymbol{v}_{\text{Earth}}(t_0)\|_2 + \|\boldsymbol{v}_2 - \boldsymbol{v}_{\text{Didymos}}(t_f)\|_2$$

where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are solutions to the Lambert problem with time of flight $\texttt{tof} = t_f - t_0$.

(d) (6 pt) Make a porkchop plot for the *outbound* trajectory (from Earth to Didymos) during the time-window from $t_{\text{ref}} + 0\,\text{day}$ to $t_{\text{ref}} + 1000\,\text{day}$, with a grid of 5-day intervals (i.e., the spacecraft needs to leave no earlier than $t_{\text{ref}} + 0$ and arrive to Didymos no later than $t_{\text{ref}} + 1000$).
Tips for porkchop plot:

- set `cw = false` (we only want prograde transfers)
- within the double for-loop of initial time t_0 and final time t_f, skip computation if $t_f - t_0 \leq +100$ days (as we expect the transfer to be very expensive anyway)

(e) (4 pt) Plot a 3D trajectory corresponding to a local minimum from your porkchop plot in (d). In your plot, also include the orbits of Earth and Didymos, as well as the position of Earth at the time of departure and the position of Didymos at the time of arrival.