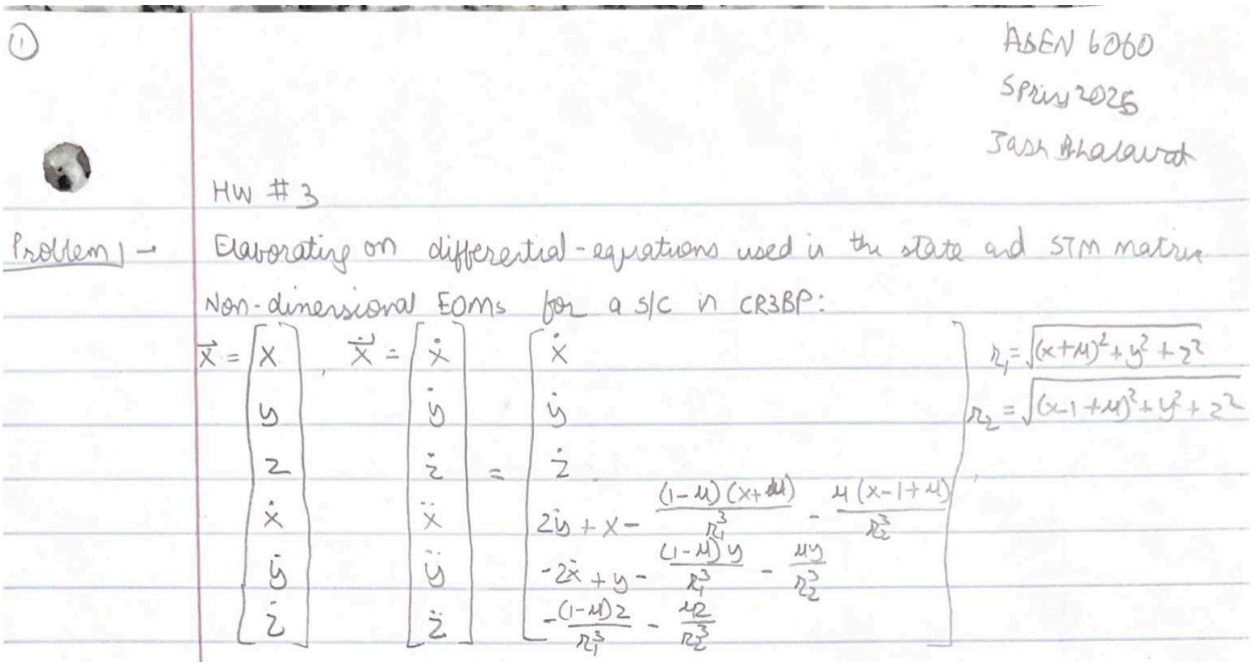
ASEN 6060 - HW 3
Spring 2025
Jash Bhalavat

Problem 1

The state vector and the state transition matrix (STM) can be augmented to create a 42x1 vector that can be integrated into a 1st order differential equation simultaneously. Firstly, the state vector differential equation is as follows:



① ASEN 6060 Spring 2025 Jash Bhalavat

HW #3

Problem 1 – Elaborating on differential-equations used in the state and STM matrix

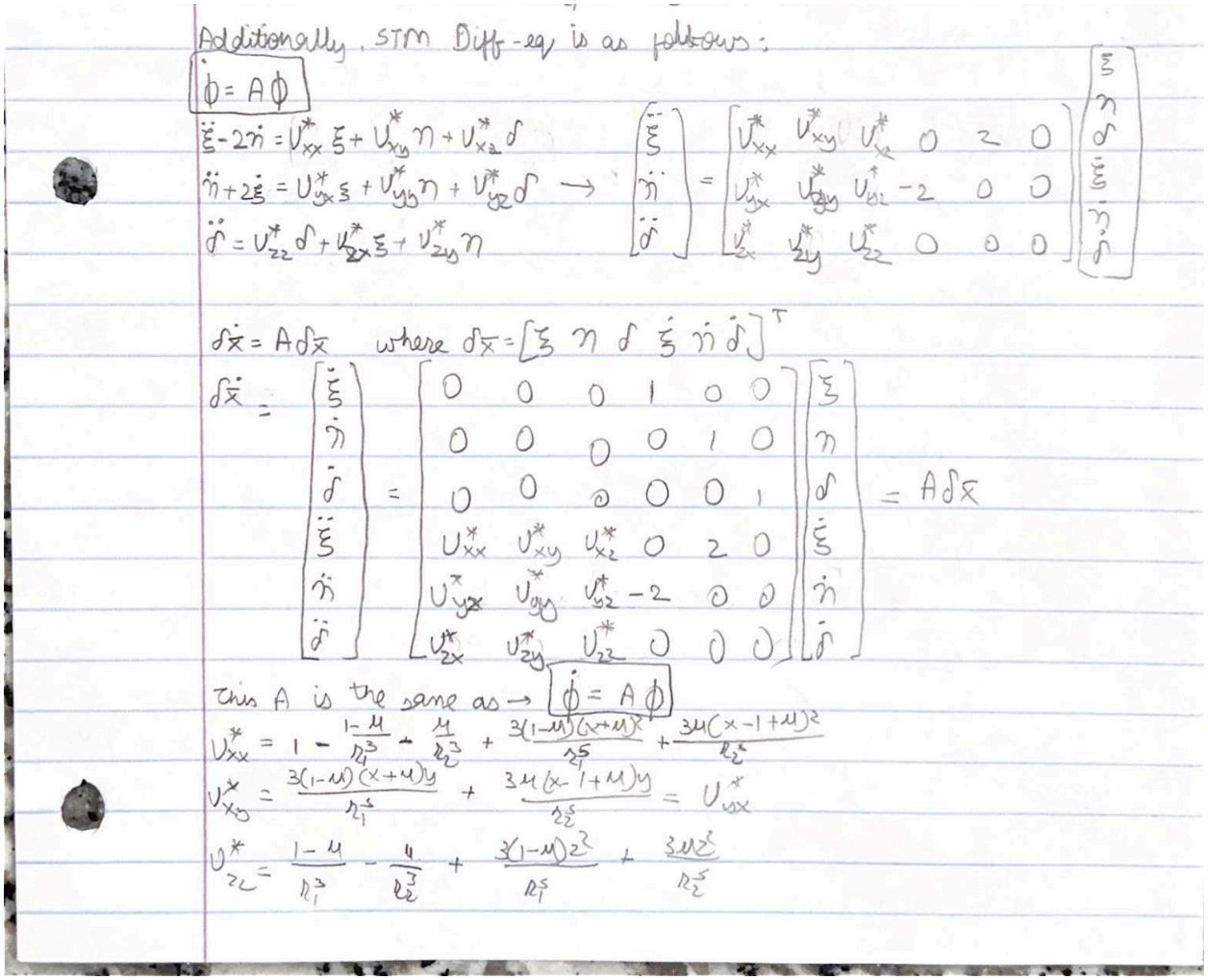
Non-dimensional EOMs for a s/c in CR3BP:

$$\vec{X} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \quad \dot{\vec{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ 2\dot{y} + x - \frac{(1-\mu)(x+\mu)}{r_1^3} - \frac{\mu(x-1+\mu)}{r_2^3} \\ -2\dot{x} + y - \frac{(1-\mu)y}{r_1^3} - \frac{\mu y}{r_2^3} \\ -\frac{(1-\mu)z}{r_1^3} - \frac{\mu z}{r_2^3} \end{bmatrix}$$

$$r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2}$$

$$r_2 = \sqrt{(x-1+\mu)^2 + y^2 + z^2}$$

Then, the STM differential equation:



This is then implemented in Matlab in a function called CR3BP_full() that inputs the state vector and the transition matrix along with the system mass ratio and outputs the time derivative of the state vector along with the time derivative of the STM using the differential equations shown above. The matlab function is shown below:

(The helper functions used throughout this homework assignment can be found in the appendix at the end of this document)

```
function state_phi_dot = CR3BP_full(state_phi, mu)
    % Full state vector and state transition matrix differential equation
    % Inputs:
    % state_phi - Augmented state vector and STM [42x1]. The state vector -
    % [x0, y0, z0, x0_dot, y0_dot, z0_dot]. The STM - is 6x6 with each
    % element described as - phi_ij = dxi(tf)/dxj(t0). The phi matrix is
    % reshaped such that all the rows are concatenated vertically. For
    % example -
    % phi_mat = [phi11, phi12, phi13, ..., phi16;
    %            [phi21, phi22, phi23, ..., phi26;
    %            ...
    %            [phi61, phi62, phi63, ..., phi66]
    % becomes
    % phi_row = [phi11, phi12, ..., phi16, phi21, phi22, ..., phi66]'
    %
    % mu - system mass ratio [-]
    %
    % Output
    % state_phi_dot - Augmented state vector dot and STM_dot [42x1]. The
    % augmentation and reshaping scheme remains the same as the input.

    x = state_phi(1);
    y = state_phi(2);
    z = state_phi(3);
    xdot = state_phi(4);
    ydot = state_phi(5);
    zdot = state_phi(6);

    r1 = sqrt((x + mu)^2 + (y)^2 + (z)^2);
    r2 = sqrt((x - 1 + mu)^2 + (y)^2 + (z)^2);

    state_dot(1, 1) = xdot;
    state_dot(2, 1) = ydot;
    state_dot(3, 1) = zdot;

    state_dot(4, 1) = 2*ydot + x - (1 - mu)*(x + mu)/(r1^3) - mu * (x - 1 +
mu)/(r2^3);
    state_dot(5, 1) = -2*xdot + y - (1 - mu)*y/(r1^3) - mu*y/(r2^3);
    state_dot(6, 1) = -(1 - mu)*z/(r1^3) - mu*z/(r2^3);

    % Calc pseudo-potentials
    uxx = u_xx(mu, [x, y, z]);
    uyy = u_yy(mu, [x, y, z]);
    uxy = u_xy(mu, [x, y, z]);
    uzz = u_zz(mu, [x, y, z]);
    uxz = u_xz(mu, [x, y, z]);
    uyz = u_yz(mu, [x, y, z]);
```

```
    U_mat = [uxx, uxy uxz; uxy, uyy uyz; uxz uyz uzz];
    Omega = [0 2 0; -2 0 0; 0 0 0];
    A = [zeros(3), eye(3);
        U_mat, Omega];

    % Get only the phi elements into a row
    phi_row = state_phi(7:end);

    % Converting phi to matrix
    phi_mat = reshape(phi_row, [6,6])';

    % Get phi_dot
    phi_dot_mat = A * phi_mat;

    % Convert back to row
    phi_dot_row = reshape(phi_dot_mat', [36,1]);

    % Augment state and phi (in row form)
    state_phi_dot = [state_dot; phi_dot_row];

end
```

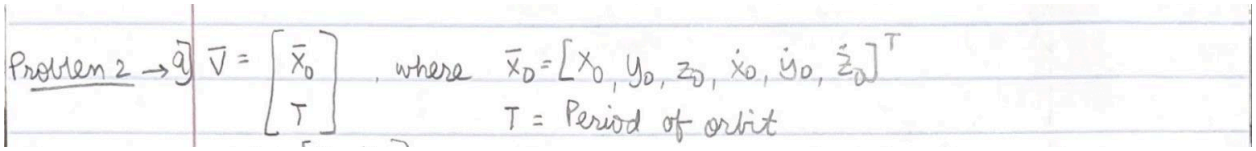
In order to implement this differential equation, Matlab's ODE113() numerical integrator is used based on feedback from HW 1. Additionally, this documentation helps choose an ODE solver and it suggests that ODE113() is more efficient than ODE45() at problems with stringent tolerances (like the CR3BP) or when the ODE function is expensive to evaluate (like the CR3BP non-linear equations). ODE113() has options to set relative and absolute tolerances. The relative tolerance roughly controls the number of correct digits in the solution greater than the absolute tolerance. The absolute tolerance dictates the lower bound of the solution. ODE113() computes the local error at each time step and this value has to be less than either the absolute or relative tolerances. Standard double precision is accurate up to 16 decimal places. So, 1e-16 is the lowest possible tolerance when operating with standard data types in Matlab. For the rest of this homework assignment, answers will be given till 13 decimal points. Therefore a tolerance of 5e-14 should suffice. Additional experimentation was done throughout this homework assignment and this tolerance strikes a balance between reasonable computational time and precise state vectors and orbital properties (such as Jacobi constant).

Here’s an example of how to implement the aforementioned CR3BP_full() function:

```
% Truncated system mass ratio for demonstration
mu = 0.0122;

% Set tolerance for numerical integrator
TOL = 5e-14;

% Set options for ode113
options = odeset('RelTol', TOL, 'AbsTol', TOL);

% Arbitrary initial state for demonstration
state0 = [1, 0, 0, 1, 0, 0]; % [m, m/s]
phi0 = eye(6); % STM matrix evaluated at initial condition is identity
phi0_row = reshape(phi0, [6, 6]);

state_phi_0 = [state0; phi0_row];

t = 1; % Arbitrary period for demonstration

[t_out, state_out] = ode113(@(t,state)CR3BP_full(state, mu), [0, t], state_phi_0, options);
```

Problem 2 - Part a

Firstly, the free variable is defined as follows:



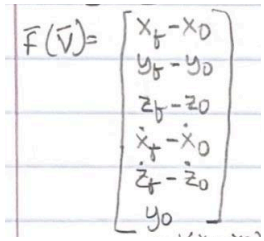
The goal of the corrections single shooting problem is to find a periodic orbit. This means that the trajectory's initial state and last states are the same. So, the constraint vector can just be found by subtracting the initial state from the final state. But, this doesn't conserve the Jacobi constant. So, one continuity constraint is removed (specifically, $\dot{y}_f - \dot{y}_0$). This places the initial state on the x-axis. When comparing the initial and final Jacobi constants:

$$2U_0^* - \dot{x}_0^2 - \dot{y}_0^2 - \dot{z}_0^2 = 2U_f^* - \dot{x}_f^2 - \dot{y}_f^2 - \dot{z}_f^2$$

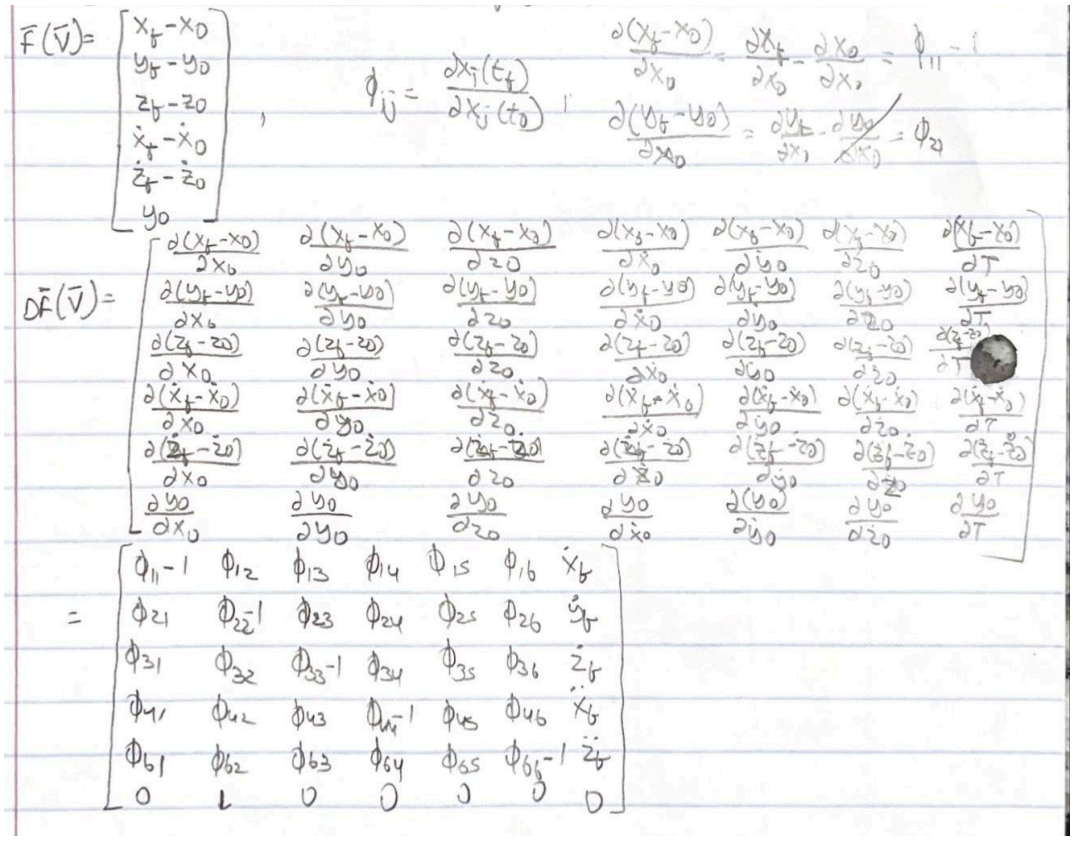
All elements cancel except the velocity in the y direction:

$$\dot{y}_0^2 = \dot{y}_f^2$$

This entails that the magnitudes of these elements must be the same (not necessarily the sign). In order to complete the constraint vector however, y_0 must be 0 since the initial condition starts at the x-axis. This is added to the constraint vector as follows:



Then, the DF matrix is computed as follows:



This single shooting is implemented in Matlab as a function called gen_3d_periodic_orbit_single_shooting() which has inputs of an initial guess for the free variable vector, the system parameters (which contain the system mass ratio) and a plot argument for the plot of constraint vector norm. Firstly, the initial state (first 6 variables of the free variable vector) is used to non-linearly propagate a trajectory (also using the period which is the 7th variable of the free variable vector). The final state of this trajectory is used to calculate the initial norm of the constraint vector (F). All of this is used within a while loop to compute the correction. There are two conditions for exiting the while loop:

- Norm of the constraint vector is below 5e-14.
    - This is considered as the “zero” threshold. Similar to problem 1, this threshold is not too strict so as to computationally strain the machine and also precise enough for 13 decimal place initial vectors presented later in the homework.
- If the while loop runs for more than 50 loops.
    - This is to ensure that if there is an error within the loop, the while loop doesn’t go on forever.

The while loop's flow can be summarized as follows:

- The while loop checks if the norm of the current constraint vector is within the threshold or if the counter has exceeded the maximum possible value (50). If either of this is true, the while loop stops.
- An initial state and STM is required to propagate the full CR3BP (including the state vector and STM). The initial state can be found from the first 6 free variables for the current free variable vector. The initial STM is an identity matrix (since, $\phi(t_0, t_0) = I_{6x6}$)
- These are augmented and fed into the CR3BP_full() function that's elaborated on in problem 1 (the 1st order differential equation for the state and the STM).
- The last row of this state+STM output is used to calculate the current constraint vector and the DF matrix.
- Then the next free variable vector is found using the following equation:

$$\bar{V}_{i+1} = \bar{V}_i - D\bar{F}(\bar{V})^T * (D\bar{F}(\bar{V}) * D\bar{F}(\bar{V})^T)^{-1} * \bar{F}$$

- The norm of the constraint vector is then calculated and stored and the counter iteration goes up by 1. Then the loop ends.

After the while loop has terminated, $\dot{y}_0^2 - \dot{y}_f^2$ is printed to manually make sure that they are "close enough". Since they are not a direct member of the constraint vector, it is not expected that these values should fall within the strict tolerance of the constraint vector. Nonetheless, a manual check is necessary to ensure that the algorithm produces a "valid" result. This is used more as a sanity check rather than a scientific test. Lastly, a plot of the norm of F (log scale) is plotted if the input argument is true.

Here is the implementation in Matlab:

```
function V_soln = gen_3d_periodic_orbit_single_shooting(V0, system_params,
plot_input)
    % Script to compute a general three-dimensional periodic orbit via single
shooting
    % Inputs
    % V0 - initial guess for a free variable vector
    % statef_V0 - final state when V0 is used as initial guess using CR3BRP
    % EOMs
    % system_params - system parameters
    %
    % Output
    % V_soln - free variable vector corresponding to a solution

    % Get mass ratio of system
    mu = system_params(1);

    % Set tolerance for numerical integrator and constraint vector
    TOL = 5e-14;

    % Set options for ode113
    options = odeset('RelTol', TOL, 'AbsTol', TOL);

    % Propagate V0 non-linear CR3BP EOMs
    [tout, xout] = ode113(@(t, state)CR3BP(state, mu), [0 V0(end)], V0(1:6),
options);

    % Final final variables using V0
    statef_V0 = xout(end,:);

    % Period is a free variable
    T = V0(end);

    % Initialize constraint vector norm
    F_norm(1) = norm(F(V0(:,1), statef_V0));

    % Matrix of all free variable vectors
    V(:,1) = V0;

    % While loop params
    counter = 1;
    counter_max = 50;

    % While loop to reduce F_norm
    while ((F_norm(counter) > TOL) && (counter < counter_max))
        phi0 = reshape(eye(6), [36, 1]); % Initial phi is identity
        state0 = [V(1:6,counter); phi0];
```

```
        % Propagate full state and STM
        [t_out, state_out] = ode113(@(t,state)CR3BP_full(state, mu), [0,
V(7,counter)], state0, options);
        statef = state_out(end, :)';
        state_dot = CR3BP_full(statef, mu);
        F_i = F(state0, statef);
        DF_i = DF_mat(statef, state_dot);

        % Find V_i+1
        V(:,counter+1) = V(:,counter) - DF_i' * inv(DF_i * DF_i') * F_i;

        % Calculate F_norm and update counter
        F_norm(counter+1) = norm(F_i);
        counter = counter + 1;
    end

    if plot_input
        figure()
        plot([1:counter], F_norm, '-o', 'LineWidth', 2)
        yscale log
        grid on
        xlabel("Iterations")
        ylabel("F Norm")
        title("Constraint Vector Norm for each Iteration")
        hold on
        tol_yline = ones([counter,1])*TOL;

        plot([1:counter], tol_yline, 'red', 'LineWidth', 2)
        hold off
        legend("Norm", "Threshold")
    end

    print_out = sprintf('Difference in y_dot_0^2 and y_dot_f^2 - %d', V0(5)^2
- V(5,end)^2);
    disp(print_out)

    V_soln = V(:,end);

end
```

```
function out = F(state0, statef)
    % Modified Constraint Vector
    out = [statef(1) - state0(1); % x0
           statef(2) - state0(2); % y0
           statef(3) - state0(3); % z0
           statef(4) - state0(4); % x0_dot
           statef(6) - state0(6); % z0_dot
           state0(2)]; % y0
end
```

```
function out = DF_mat(state, state_dot)
    % Modified constraint DF matrix

    % Convert STM to matrix from row vector
    phi_row = state(7:end);
    phi_mat = reshape(phi_row, [6,6])';
    % Subtract identity and extract all rows except 5th row
    phi_minus_I = phi_mat - eye(6);
    phi_mod = [phi_minus_I(1:4, :); phi_minus_I(6,:)];

    % Grab states - x_dot, ydot, zdot, xdotdot, zdotdot
    state_dot_col = [state_dot(1:4); state_dot(6)];

    out = [phi_mod, state_dot_col; 0 1 0 0 0 0 0];
end
```

Problem 2 - Part b

For the initial guess, firstly, the A2D matrix is calculated for L1.

$$\begin{bmatrix} \dot{\xi} \\ \dot{\eta} \\ \ddot{\xi} \\ \ddot{\eta} \end{bmatrix} = \begin{bmatrix} 0_{2x2} & I_{2\times 2} \\ U^*_{XX} & \Omega \end{bmatrix} \begin{bmatrix} \xi \\ \eta \\ \dot{\xi} \\ \dot{\eta} \end{bmatrix} \longrightarrow \delta\dot{\bar{x}}_{2D} = A_{2D}\delta\bar{x}_{2D}$$

A2D = $$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 11.295188987111322 & 0 & 0 & 2 \\ 0 & -4.147594493555661 & 2 & 0 \end{bmatrix}$$

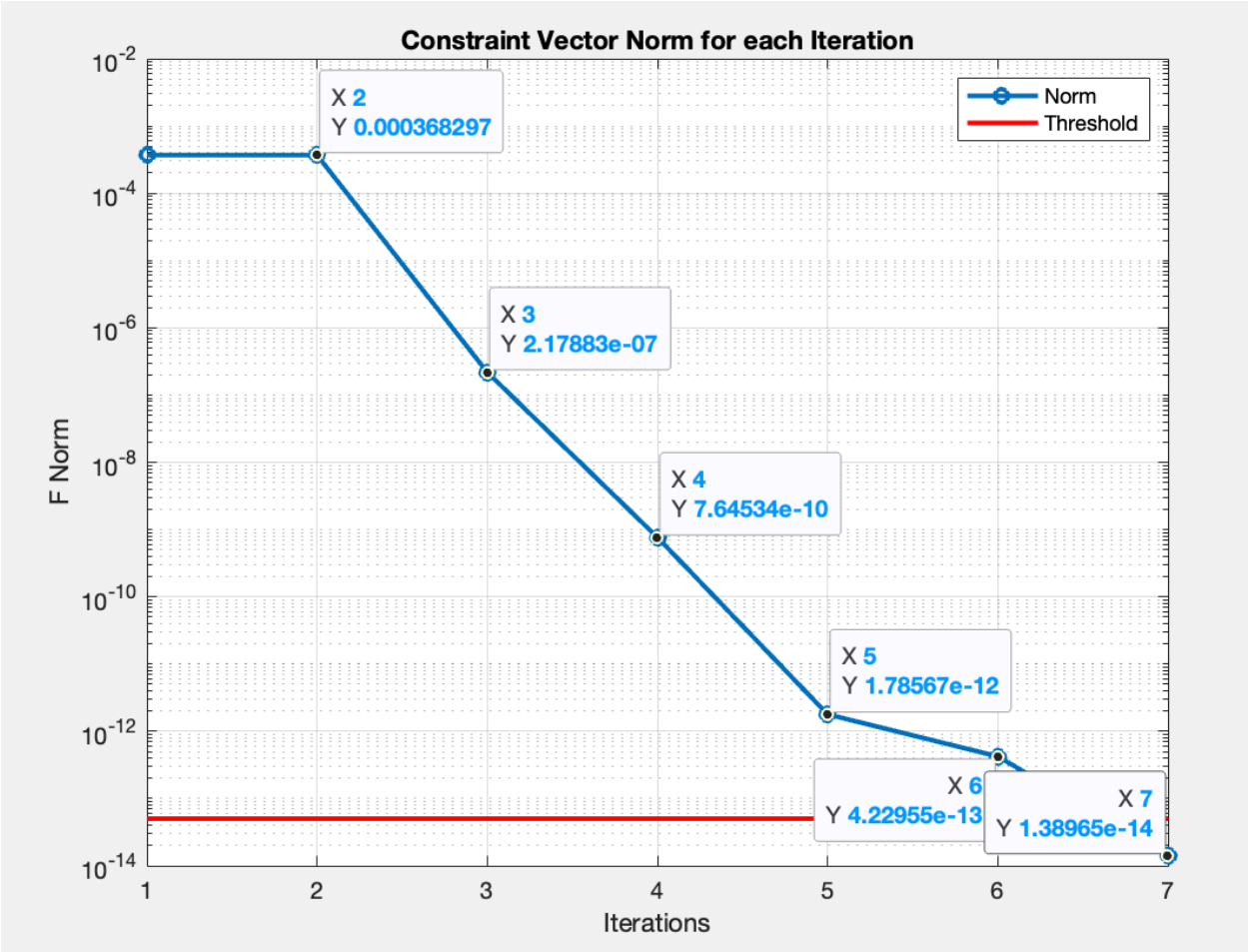
Then, using Matlab to find the oscillatory eigenvectors (corresponding to a b*i type of eigenvalue):

- Eigenvalue:
  - +/-2.3343858756070i
    - The positive eigenvalue and associated eigenvector is used for the initial guess.
- Eigenvector:
  - [-0.1057580725982 + 0i, 0.0 - 0.3793012484900i, 0.0 - 0.2468801509048i, 0.8854354770752 + 0i]

- As seen in HW2, a linear combination of the eigenvectors can be used as an initial guess for the L1 Lyapunov orbit. We will only use the real components.
  - [-0.1057580725982, 0.0, 0.0, 0.8854354770752]
- Then, the magnitude of the first two elements (position elements) are used to scale the entire vector. This scaling factor was a little bit of trial and error. The scaling factor that ended up providing the best result was 9.5e-04. The initial condition is then as follows:
  - [0.8368151317503, 0.0, 0.0, 0.0, 0.0008372273201, 0.0]
    - Note that the initial condition crosses the x-axis as intended for the modified vector constraint.
- The orbit period is $2\pi/imag(\lambda)$ = 2.6915795596757

So, the initial guess for the free variable vector is:
[0.8368151317503, 0.0, 0.0, 0.0, 0.0008372273201, 0.0, 2.691579596757]^T
This is a reasonable guess because it is “sufficiently” close to the L1 point, it’s planar, and it starts at the x-axis with a non-zero initial x-velocity term.

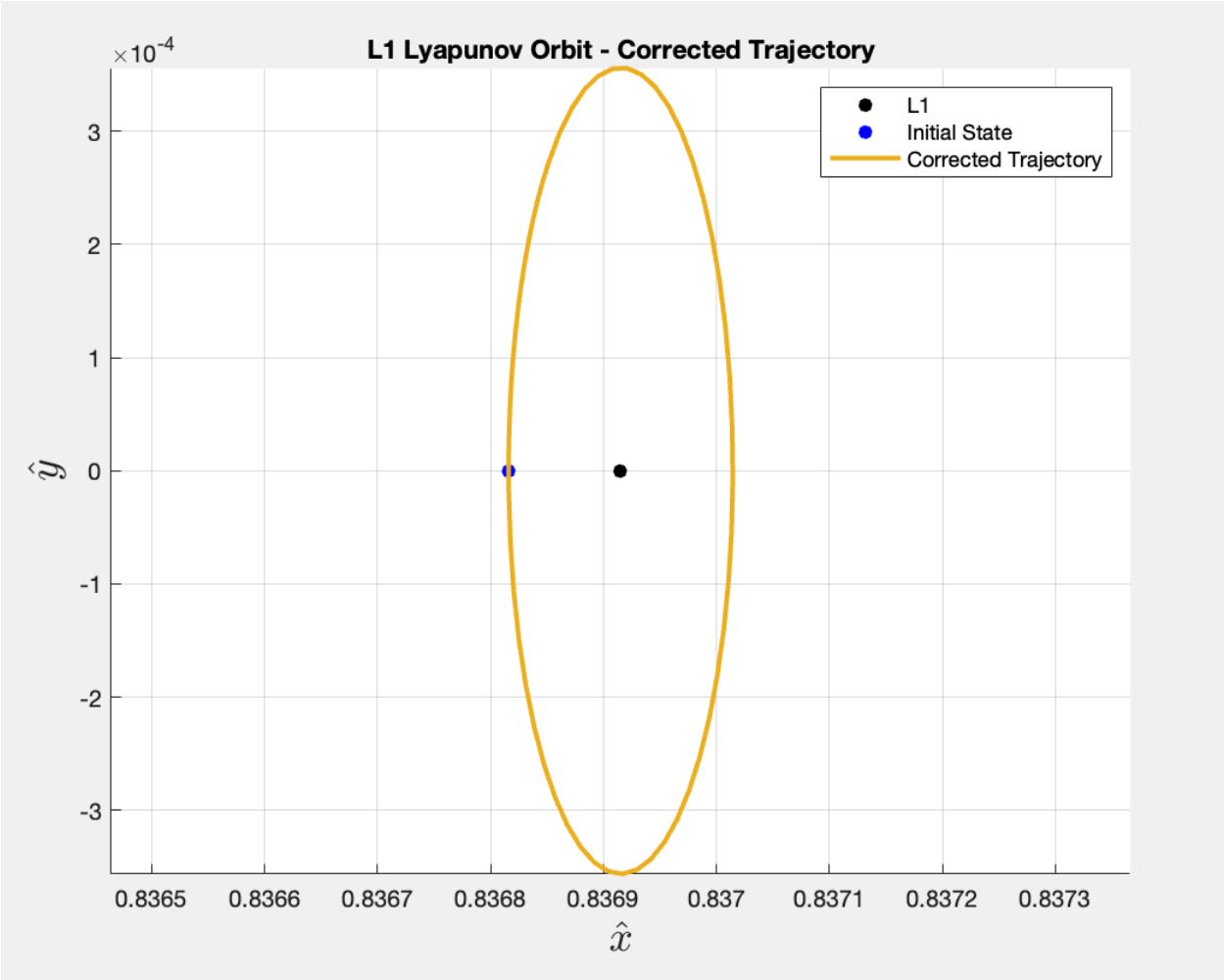
Problem 2 - Part c



The threshold is 5e-14 and as seen, the 7th iteration is below the threshold so the algorithm terminates successfully.

The quadratic trend of this plot indicates that the implementation is likely correct. Additionally, (this cannot be seen in this plot) but at no point was the DF matrix singular (this is indicated by Matlab in the terminal window) which also indicates that the DF matrix was correctly correcting the free variable vector.
The success of this implementation can be gauged based on a few criteria. Firstly, the norm of the constraint vector is below the threshold indicating that most of the continuity elements are zero. Then, the difference between the y-velocity terms is printed to the terminal and is 1.0e-8 which is sufficiently small to indicate that this corrections scheme is successful. Lastly, the corrected trajectory is plotted (in the next part) which also indicates a successful correction because the orbit (visually) starts and ends at the same point like a periodic orbit.

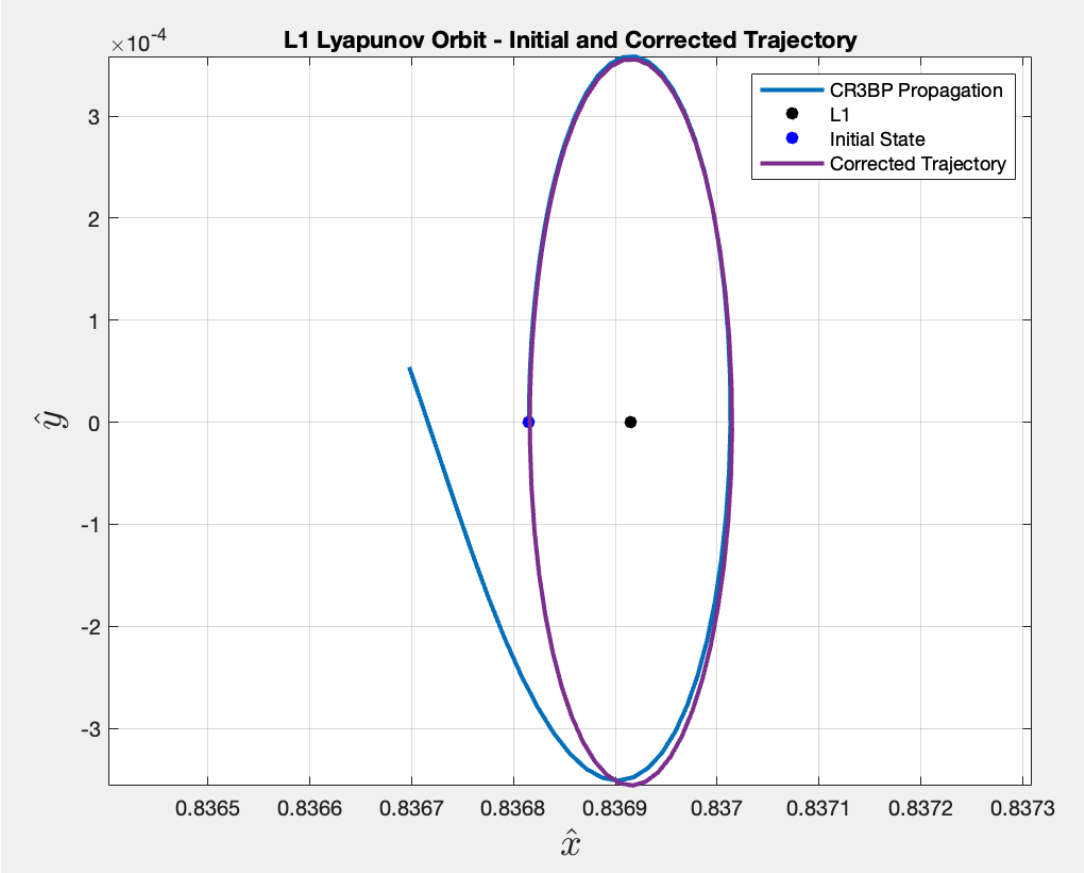
Problem 2 - Part d



Element	Initial Guess	Corrected
x_0	**0.8368151317503**	**0.8368159523850**
y_0	**0.0**	**0.0**
z_0	**0.0**	**0.0**
$\dot{x}_0$	**0.0**	**0.0**
$\dot{y}_0$	**0.0008372273201**	**0.0008309659356**
$\dot{z}_0$	**0.0**	**0.0**

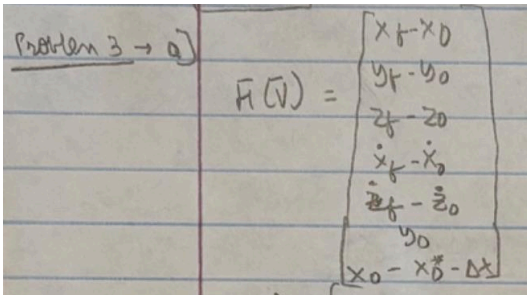
T (period)	**2.6915795596757**	**2.6915815961669**

Additional plot (out of curiosity). Comparing the corrected trajectory vs the initial guess from HW 2:



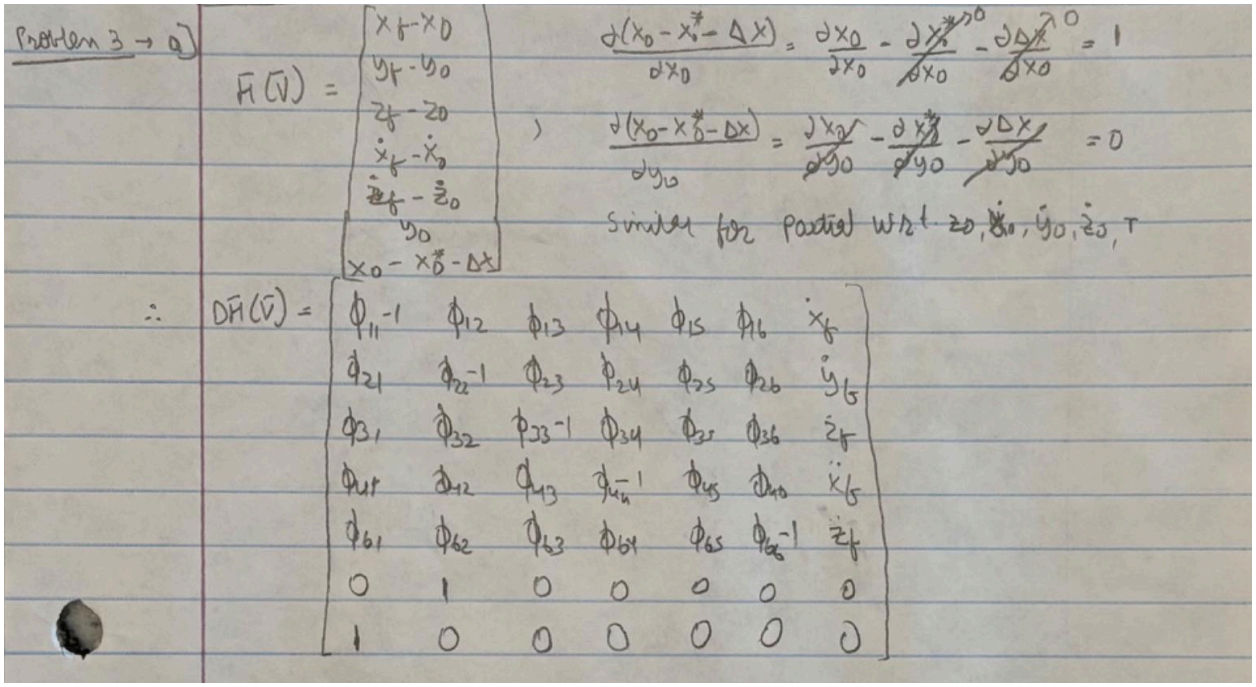
Problem 3 - Part a

Let's first define the constraint vector for the natural parameter continuation as follows:



(where x_0^* is the previously computed x_0 within the continuation and Δx is a constant).

This is then used to calculate the DH matrix as follows:



Let's talk about the implementation next. This is implemented in Matlab in a function named natural_param_continuation(). The inputs are the corrected free variable vector and system mass ratio. Firstly, the earth-moon systems' average semi-major axis is used to normalize the radius of the earth and the moon. This normalized radius is used as a termination condition (explained further below).

The initial state (first 6 variables of the corrected free variable vector) is used to non-linearly propagate a trajectory (also using the period which is the 7th variable of the corrected free variable vector). The final state of this trajectory is used to calculate the initial norm of the constraint vector (H). The initial state is also used to make sure that it's not near the boundary of either the Earth or the Moon's radius. All of this is used within a while loop to compute the correction. There are two conditions for exiting the while loop:

- If the initial state is near the Earth or Moon's surface (normalized).
    - If this is the case, we want to terminate this continuation because the trajectory will go through the moon/earth (which is almost always unwanted).
- If the while loop runs for more than 150 iterations.
    - This is to ensure that if there is an error within the loop, the while loop doesn't go on forever.
    - The reason for choosing 150 iterations is the result of trial and error. After the 150th orbit, the computation time increases a lot. This may be because the end of the orbit family has been reached. This can be confirmed by comparing the L1 Lyapunov orbit family presented here: Transfers from distant retrograde orbits to low lunar orbits - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-Lyapunov-orbits-associated-to-the-L1-and-L2_fig2_343568935 [accessed 4 Mar 2025].

The while loop's flow can be summarized as follows:

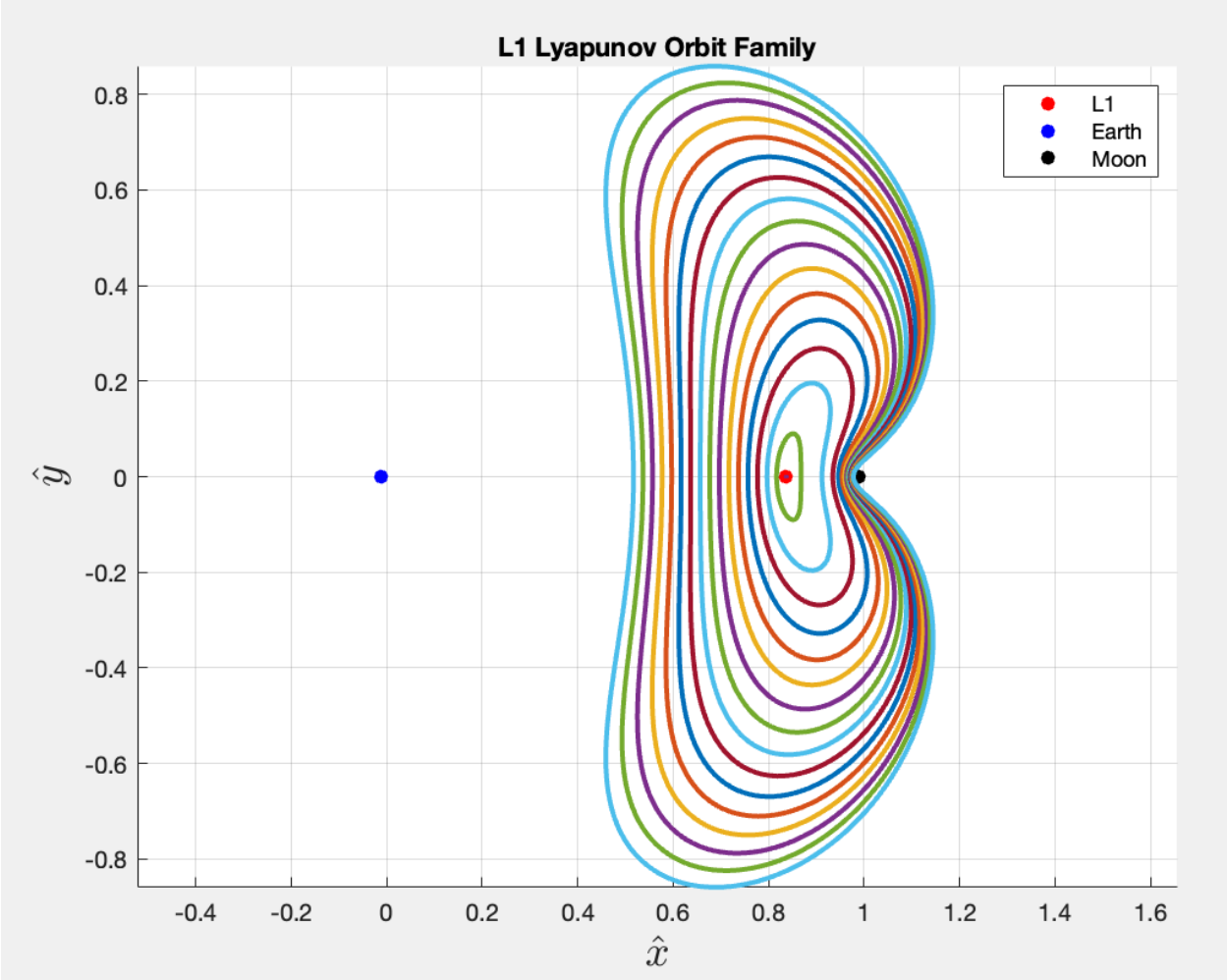
- The current free variable vector is obtained (if this is the first iteration, it's the corrected vector)
- The necessary free variable is varied using a constant delta
    - In this case it's the x0 element.
    - The delta is constant at -2e-3 because we want orbits to go away from L1 towards the earth.
- Then, the norm of the H vector is calculated.
- Another while loop is instantiated with the goal of continuing the free variable vector
    - This inner while loop has two terminating conditions
        - Norm of H vector is below 5e-14
        - Iterations are less than 20 (to avoid infinite loops)

- This while loop incrementally calculates the free variable vector that accurately continues the orbit family using the following equation:

$$\bar{V}_{i+1} = \bar{V}_i - D\bar{H}(\bar{V})^{-1} * \bar{H}$$

- The inner while loop terminates and the free variable vector is stored and the counter is reset to 1.
- When the outer while loop terminates, the whole family of free variable vectors is output.

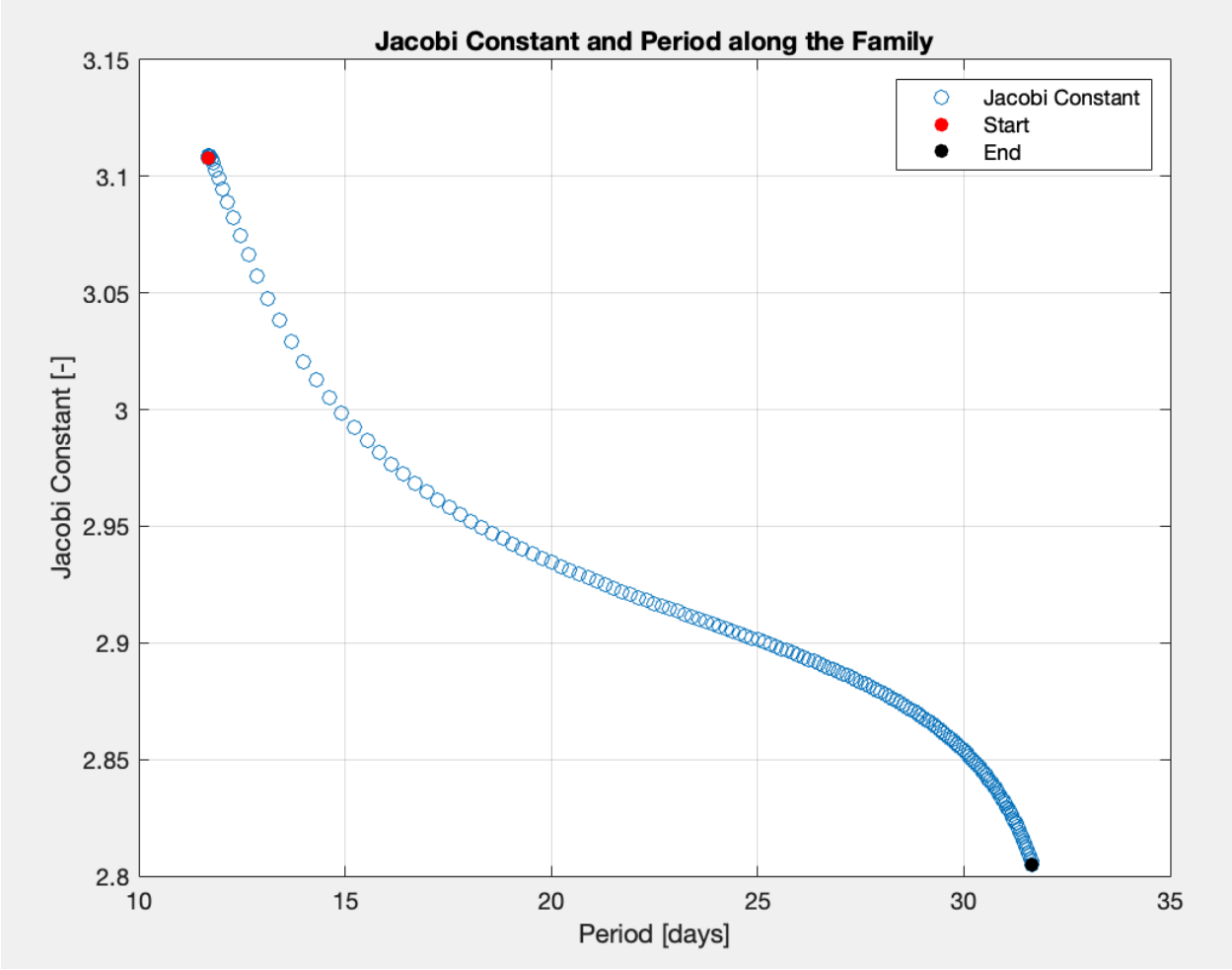
Problem 3 - Part b



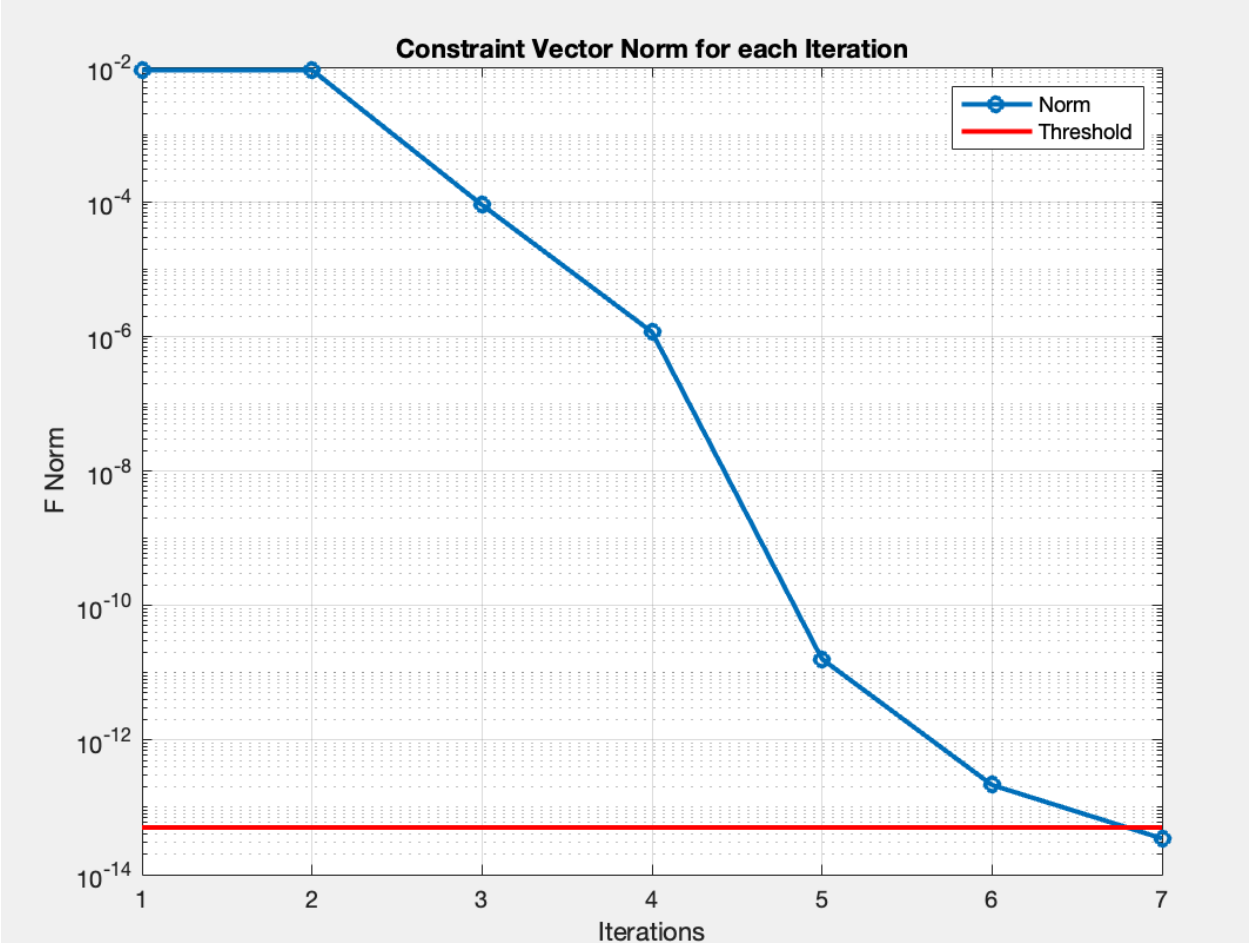
(cannot be seen in this graph, but the direction of all these orbits is prograde in the configuration space)

I don’t think this is quite the end of the family because I think there are still some orbits closer to the earth. But, my implementation saturates here. At this point, my script takes far too long to generate the next continuation within the family. But, comparing with this reference (Transfers from distant retrograde orbits to low lunar orbits - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-Lyapunov-orbits-associated-to-the-L1-and-L2_fig2_343568935 [accessed 4 Mar 2025]), it seems like the computed orbits are accurate.

Another validation can be done with a sanity check of the next plot of the jacobian as a function of the period. As the period increases, the Jacobian decreases. This relationship agrees with the fact that a decrease in the Jacobian entails more allowable motion in the CR3BP.



Problem 4



Element	Initial Guess (provided)	Corrected
x_0	0.8234	0.8234026140168
y_0	0.0	0.0
z_0	-0.026755	-0.0267277876944
$\dot{x}_0$	0.0	0.0
$\dot{y}_0$	0.1372	0.1373980193815
$\dot{z}_0$	0.0	0.0

T (period)	**2.7477**	**2.7477646185751**

The performance of the algorithm can be assessed based on the norm of the constraint vector, the norm plot, and the y-velocity difference between the initial and final states.

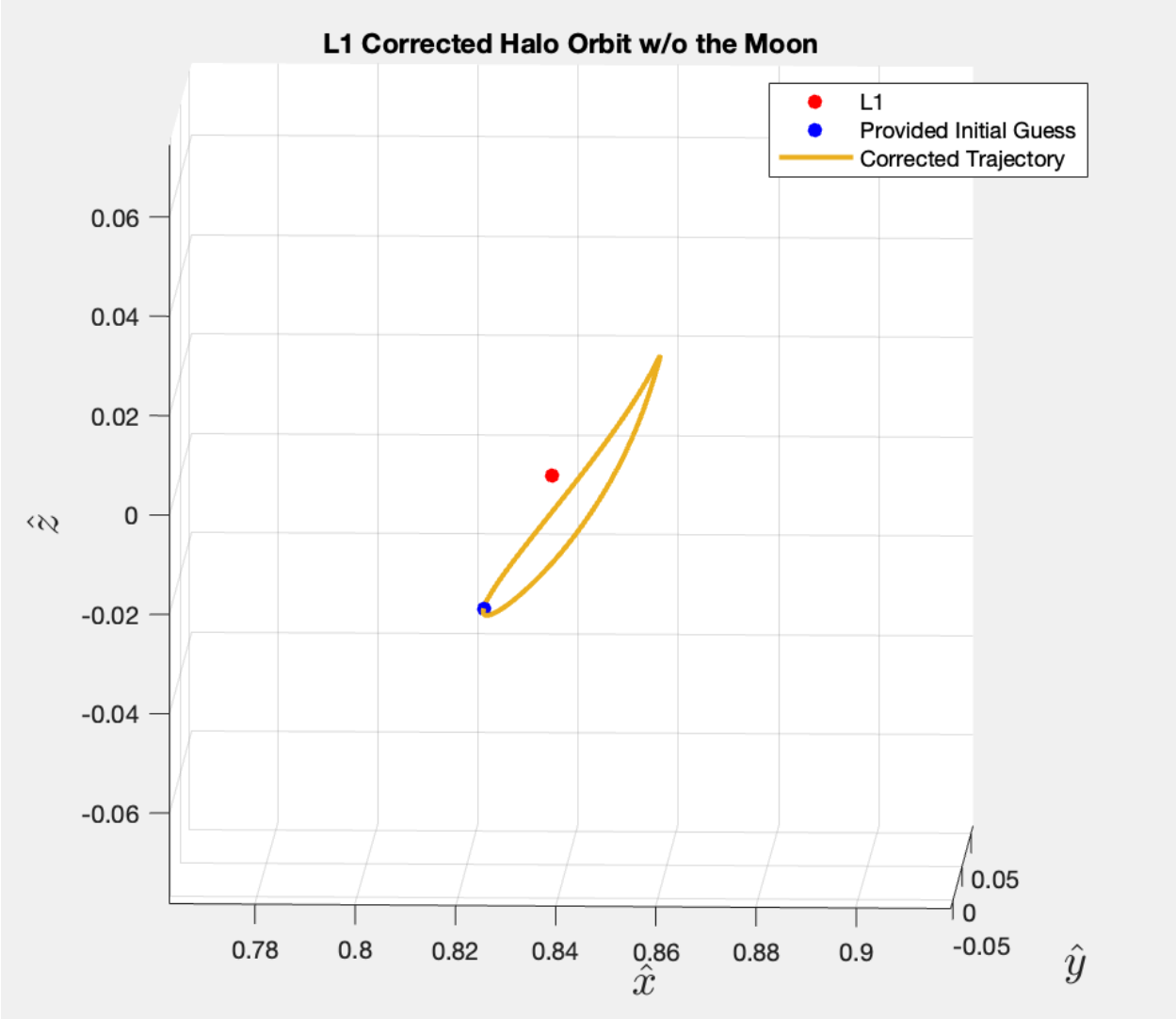
The final norm of the constraint vector is below the 5e-14 threshold which for this algorithm essentially means that it's sufficiently close to 0. This is the first criteria to ascertain the success of the single shooting algorithm.

Additionally, the constraint vector norm plot shows that it quadratically(-ish) decreases which is another indication that this algorithm has successfully executed.

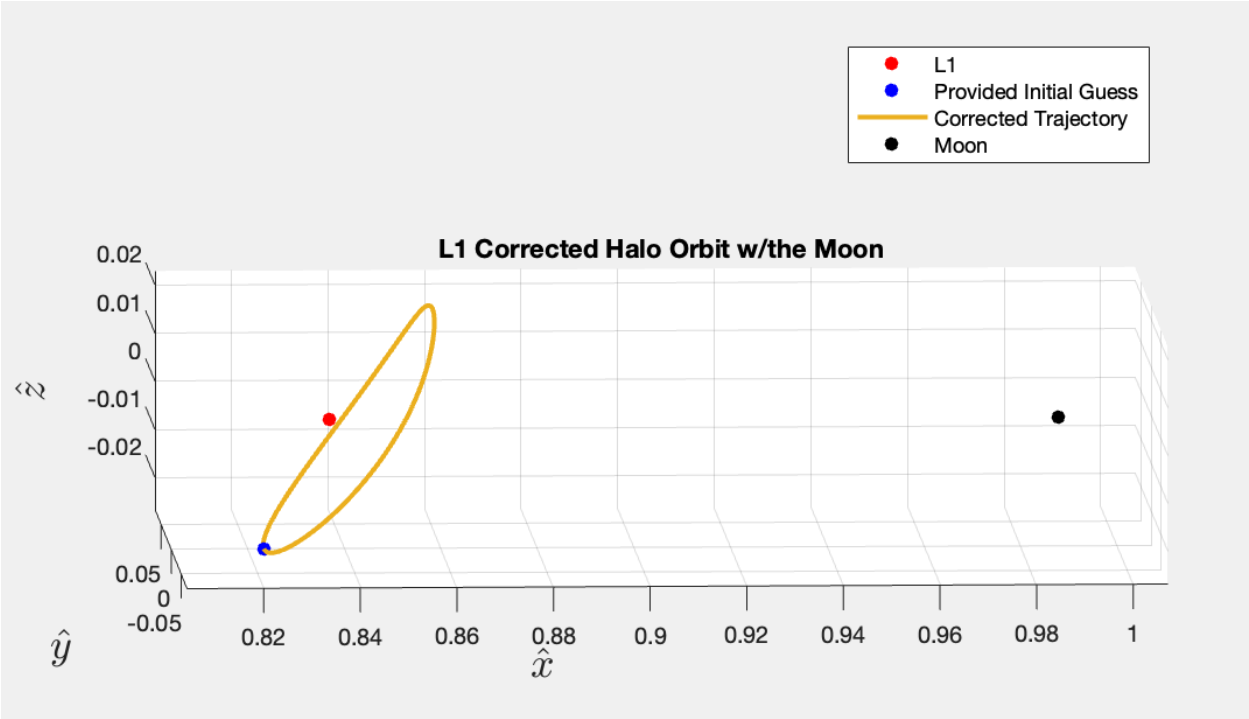
The difference between y-velocity squares at the initial and final states are also compared - $\dot{y}_0^2 - \dot{y}_f^2$ and that value is 6.0e-06. Even though this is not close to the threshold 5e-14, since this value is not being constrained via the constraint vector, a difference of 6.0e-06 is sufficient.

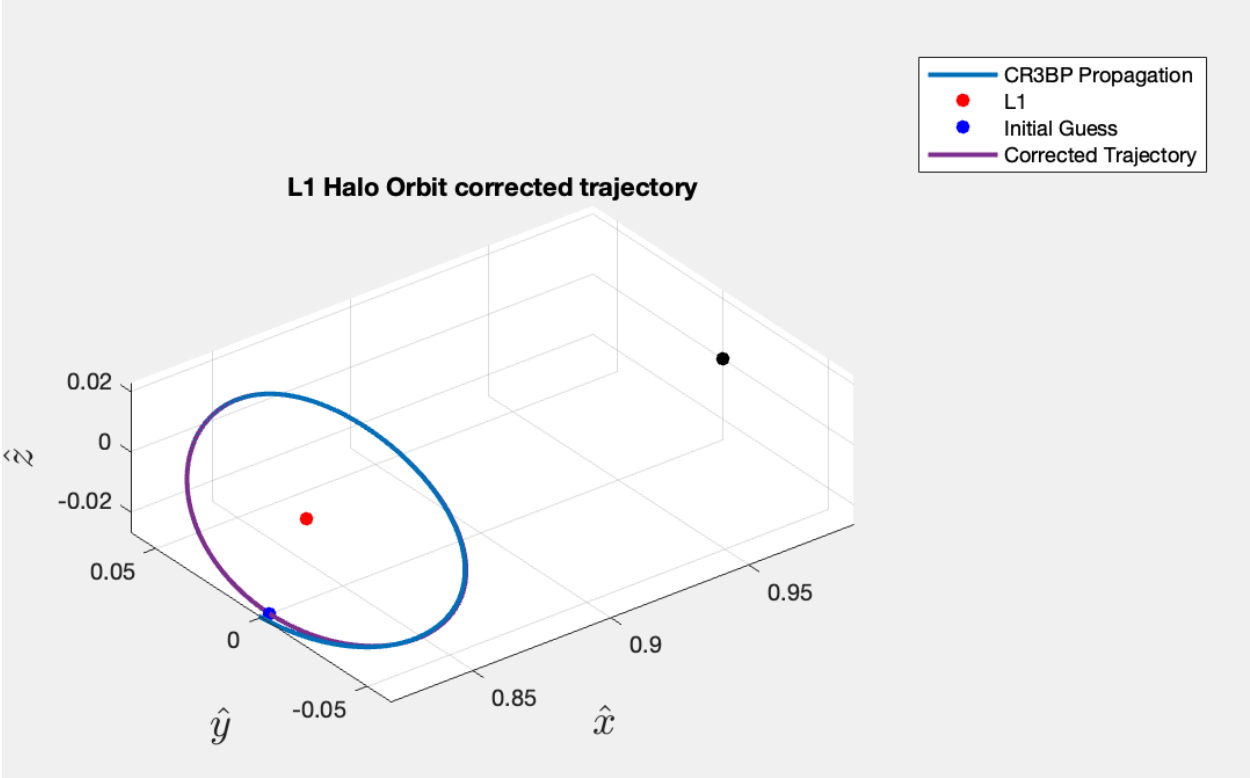
All of the above reasons lead to the conclusion that this algorithm has successfully converted the initial guess into an initial condition for an L1 halo orbit.

The plot for the corrected trajectory along with the initial guess is provided below:


L1 Corrected Halo Orbit w/o the Moon
L1
Provided Initial Guess
Corrected Trajectory
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
$\hat{z}$
0.78
0.8
0.82
0.84
0.86
0.88
0.9
$\hat{x}$
0.05
0
-0.05
$\hat{y}$

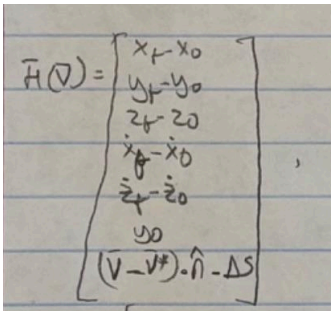
Additional orbit plots are also added below:





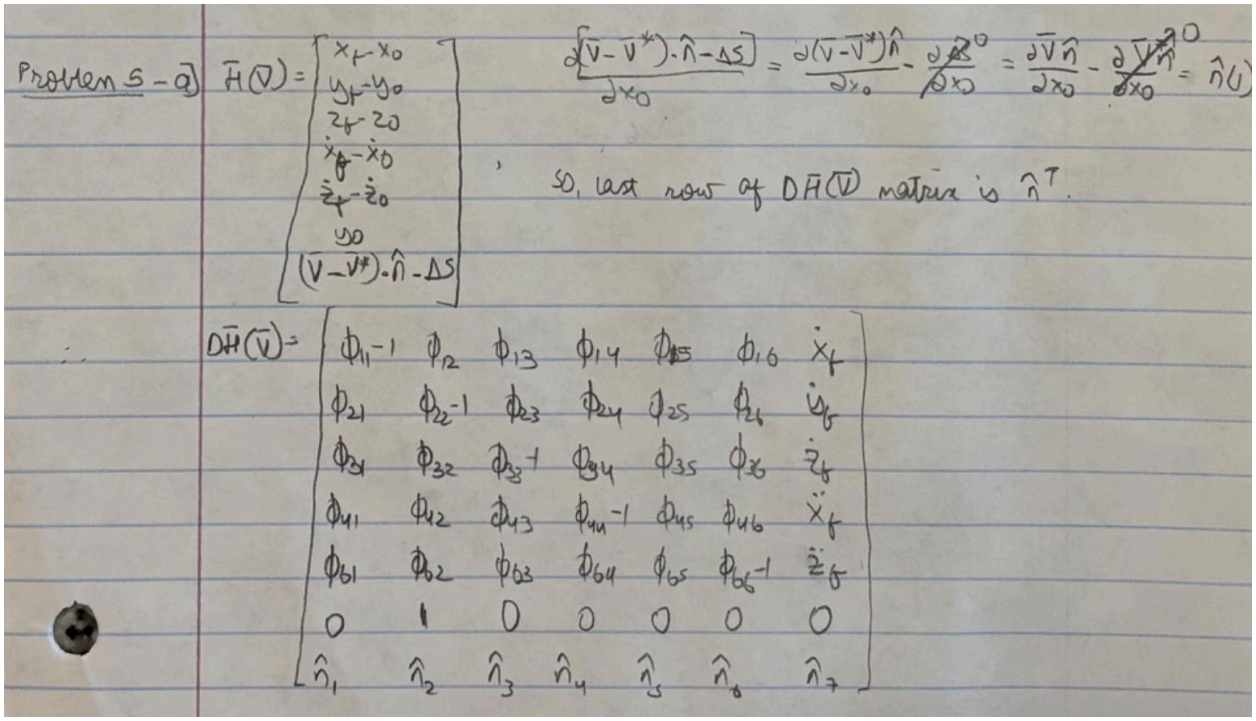
Problem 5 - part a

Let's first define the constraint vector for the pseudo-arclength continuation as follows:



(where $\bar{V}$ is the current free variable vector, $\bar{V}^*$ is the previously computed free variable vector, $\hat{n}$ is the null space of the DF matrix and Δs is a constant that steps along the family).

This is then used to calculate the DH matrix as follows:



Let's talk about the implementation next. This is implemented in Matlab in a function named pseudo_arc_length_continuation(). The inputs are the corrected free variable vector and system mass ratio. Firstly, the earth-moon systems' average semi-major axis is used to normalize the radius of the earth and the

moon. This normalized radius is used as a termination condition (explained further below).

The initial state (first 6 variables of the corrected free variable vector) is used to non-linearly propagate a trajectory (also using the period which is the 7th variable of the corrected free variable vector). The final state of this trajectory is used to calculate the initial norm of the constraint vector (H). The initial state is also used to make sure that it's not near the boundary of either the Earth or the Moon's radius. All of this is used within a while loop to compute the correction. There are two conditions for exiting the while loop:

- If the initial state is near the Earth or Moon's surface (normalized).
    - If this is the case, we want to terminate this continuation because the trajectory will go through the moon/earth (which is almost always unwanted).
- If the while loop runs for more than 2000 loops.
    - This is to ensure that if there is an error within the loop, the while loop doesn't go on forever.

The while loop's flow can be summarized as follows:

- The current free variable vector is obtained (if this is the first iteration, it's the corrected vector) and set as $\bar{V}_d$
- The $\hat{n}$ for this free variable vector is calculated.
- This is compared with the previous $\hat{n}$ to make sure these are both in the same direction. If it's in the opposite direction, then the current null space is multiplied by -1 to change its direction. This null space vector is then normalized so that its magnitude is 1.
- The current free variable vector is checked to make sure that it's not impacting either the earth's surface or the moon's surface.
- The current $\bar{V}$ is used to calculate first guess of the next free variable vector using this equation:

$$\bar{V}_0 = \bar{V}^* + \Delta s * \hat{n}$$

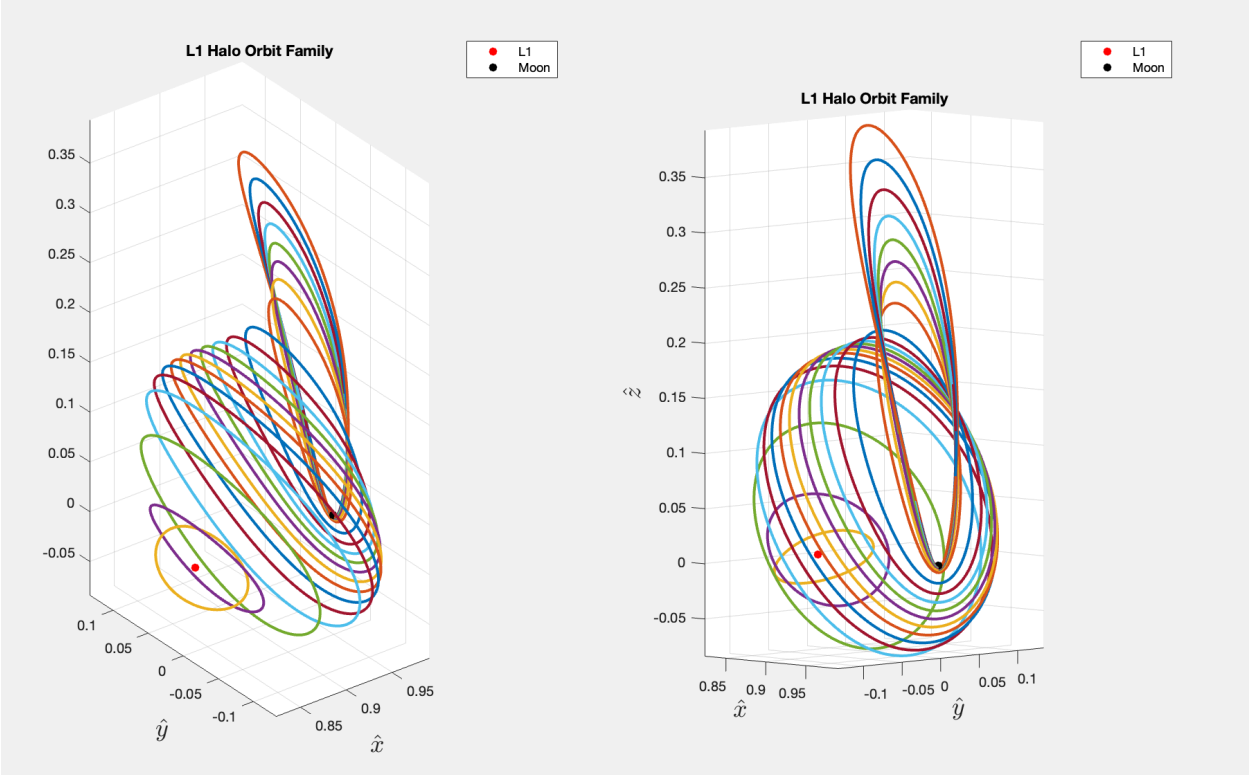
    - Δs of -1e-3 is used for the southern family and 1e-3 is used for the northern family.
- Then, the norm of the H vector is calculated.

- Another while loop is instantiated with the goal of continuing the free variable vector. The $\bar{V}_0$ calculated above is used to start this while loop and is iterated till the norm of the H vector is small enough.
    - This inner while loop has two terminating conditions
        - Norm of H vector is below 5e-14
        - Iterations are less than 20 (to avoid infinite loops)
    - This while loop incrementally calculates the free variable vector that accurately continues the orbit family using the following equation:

$$\bar{V}_{i+1} = \bar{V}_i - D\bar{H}(\bar{V})^{-1} * \bar{H}$$

- The inner while loop terminates and the free variable vector is stored and the counter is reset to 1.
- When the outer while loop terminates, the whole family of free variable vectors is output.

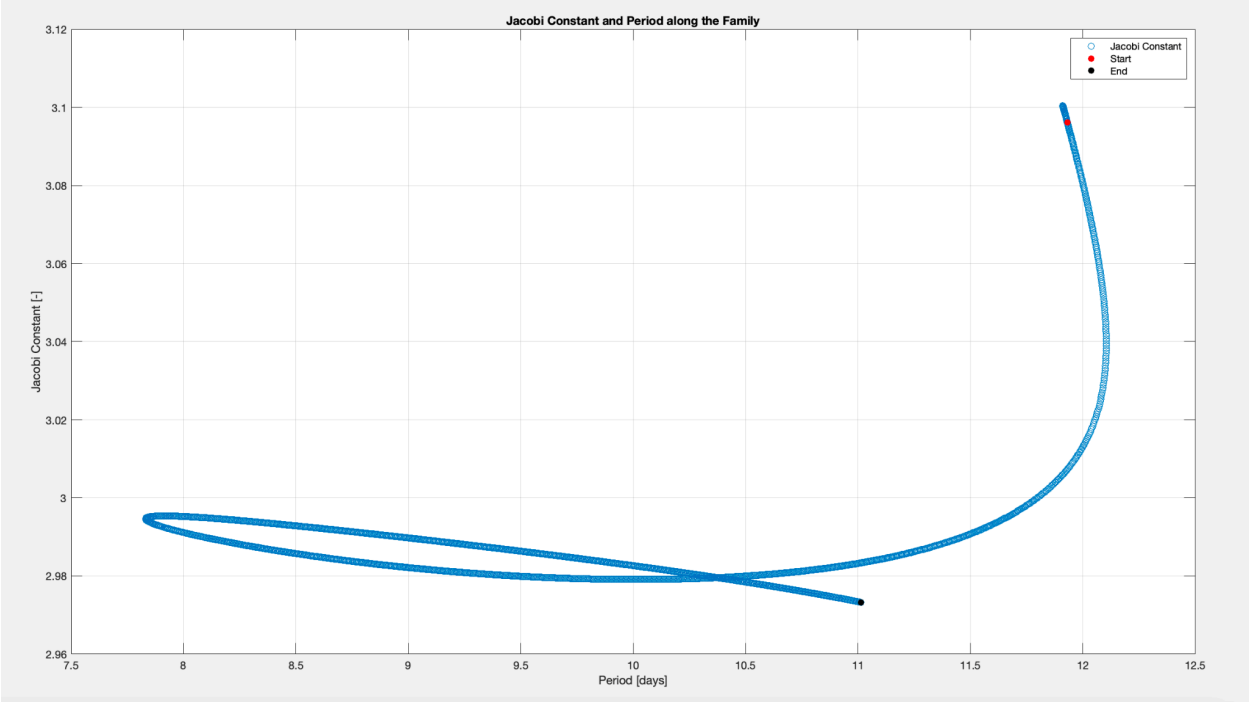
Problem 5 - part b



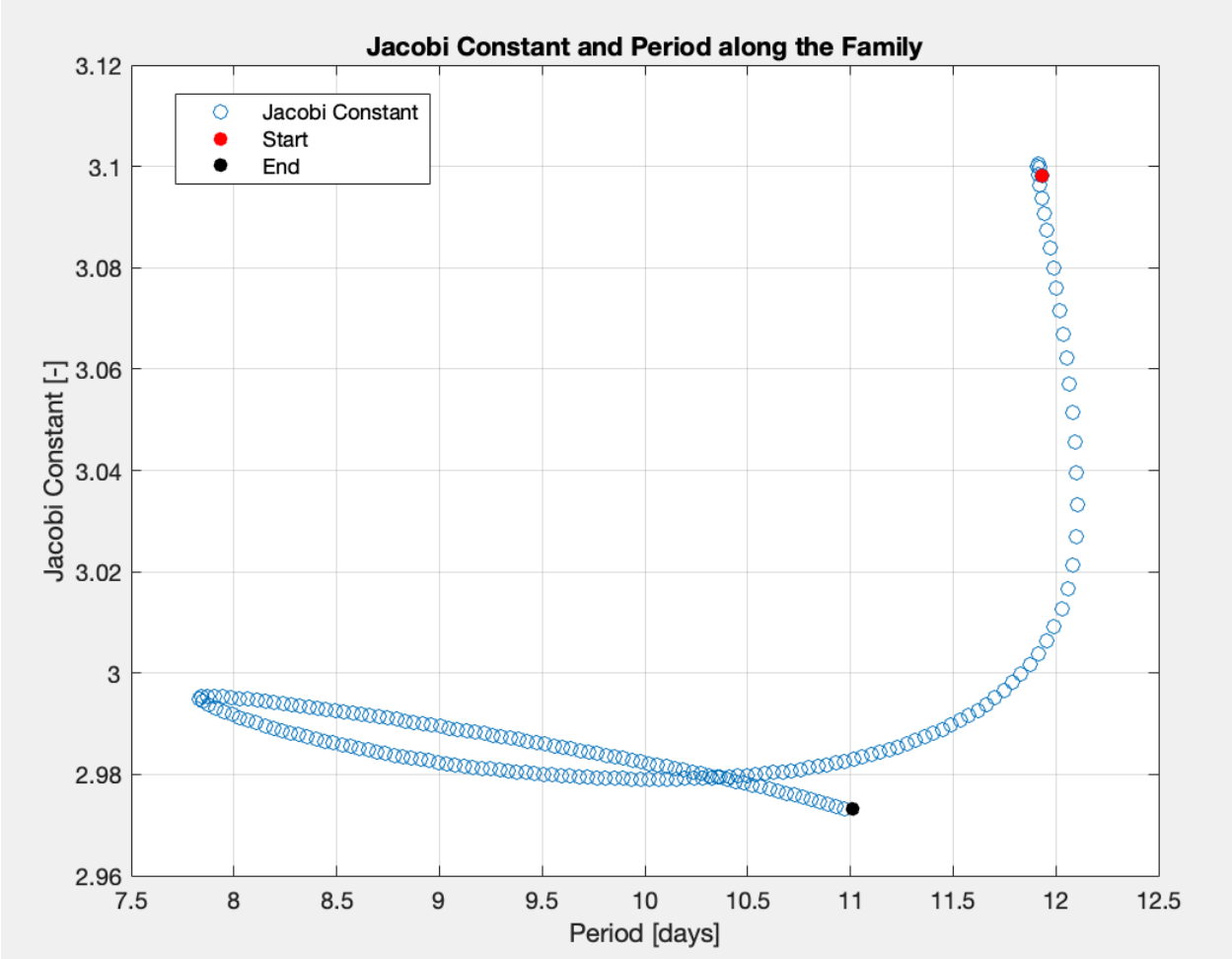
(Both plots above are the same. They're just from different perspectives to more clearly portray the family.)

This family is computed almost up to the near-rectilinear halo orbit. Also, it can be visually confirmed that this family is the southern halo orbit because the periapsis is near the southern portion of the moon.

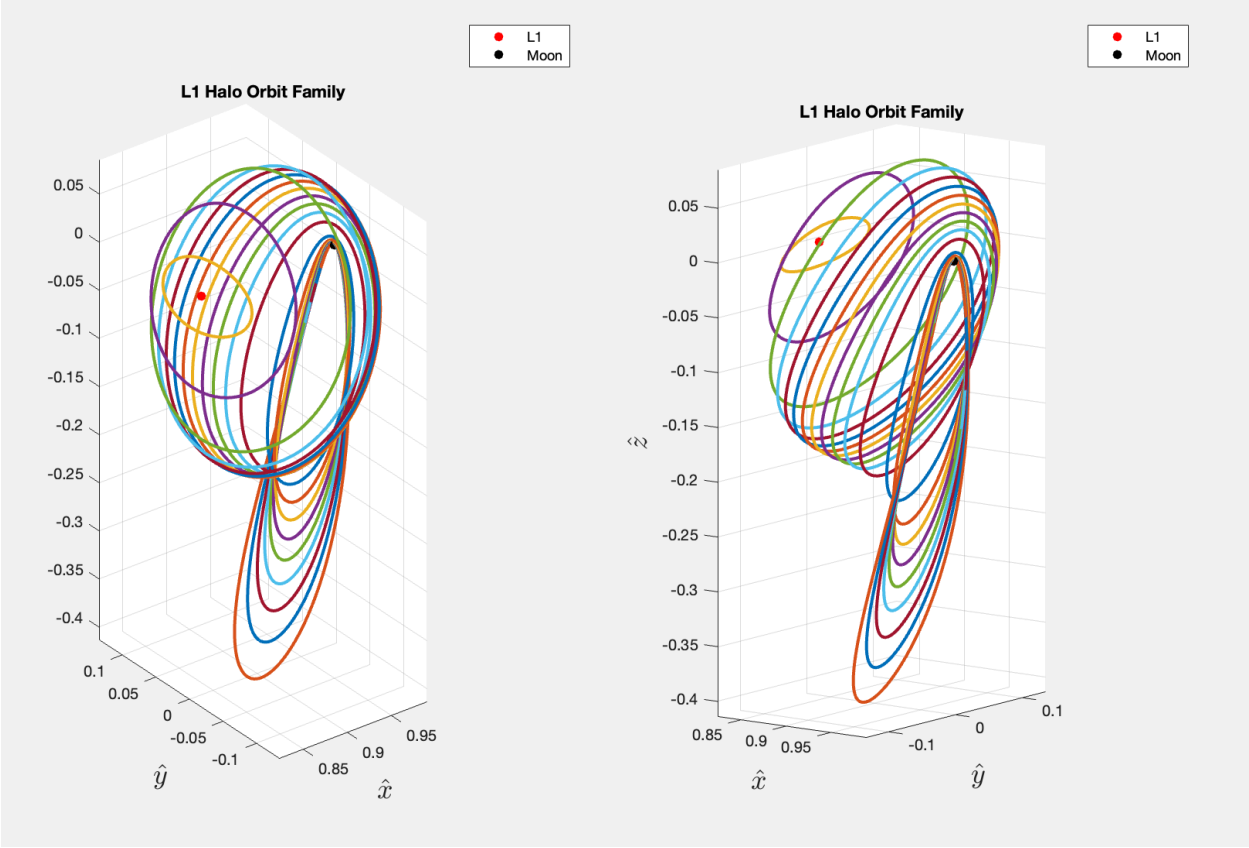
A validation of the orbit family can be found in the next plot that plots the Jacobian compared to the orbit period. The initial continuation goes towards an increasing Jacobian, but that is then reversed and it follows a smooth curve and even encounters a turning point successfully. This indicates that the pseudo-arclength continuation is implemented correctly.



Plotting all the iterations leads to a dense plot. Hence, reducing that in the plot below by plotting only every 10th iteration.


Jacobi Constant and Period along the Family
Jacobi Constant
Start
End
Jacobi Constant [-]
Period [days]
3.12
3.1
3.08
3.06
3.04
3.02
3
2.98
2.96
7.5
8
8.5
9
9.5
10
10.5
11
11.5
12
12.5

Repeating the same for the northern halo orbit family (the only difference is the step size is now 1e-3):



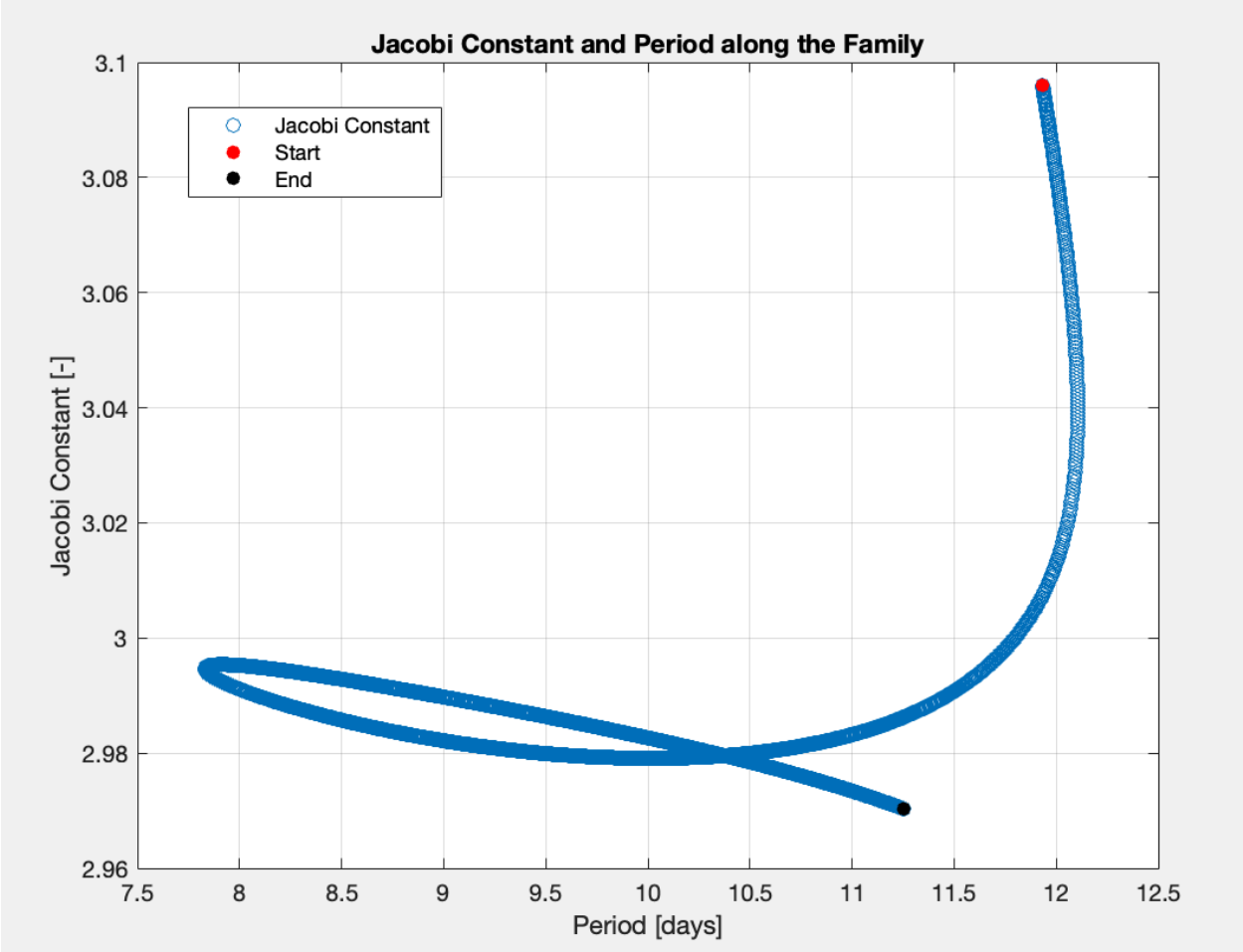

Jacobi Constant and Period along the Family
Jacobi Constant
Start
End
Jacobi Constant [-]
Period [days]
3.1
3.08
3.06
3.04
3.02
3
2.98
2.96
7.5
8
8.5
9
9.5
10
10.5
11
11.5
12
12.5

Table of Contents


```
clear; clc; close all;
```

CODE FOR HW APPENDIX (IGNORE ALL PLOTS)

Tolerances have been changed to accommodate all the code

```
% ASEN 6060 - HW 3 Code (Appendix)
% Spring 2025
% Jash Bhalavat
```

Constants

```
G = 6.67408 * 10^-11; % m3/(kgs2)
G = G / (10^9); % km3/(kgs2)

% Earth
mu_earth = 398600.435507; % km3/s2
a_earth = 149598023; % km
e_earth = 0.016708617;
mass_earth = mu_earth / G; % kg

% Moon
mu_moon = 4902.800118; % km3/s2
a_moon = 384400; % km
e_moon = 0.05490;
mass_moon = mu_moon / G; % kg

% Earth-Moon system
mass_ratio_em = mass_moon / (mass_earth + mass_moon);
m_star_em = mass_earth + mass_moon;
l_star_em = a_moon;
t_star_em = sqrt(l_star_em^3/(G * m_star_em));

mu = mass_ratio_em;
```

```
% Earth Moon system equilibrium points
[em_eq_pts, em_eq_validity] = all_eq_points(mu);

% Only looking at L1 eq point planar oscillatory modes
l1_pos = [em_eq_pts(1,:), 0];

l1_in_plane_modes = in_plane_modes(mu, l1_pos);

oscillatory_eval = l1_in_plane_modes(3);
uxx_l1 = u_xx(mu, l1_pos);
uxy_l1 = u_xy(mu, l1_pos);
uyy_l1 = u_yy(mu, l1_pos);
U_star_XX = [uxx_l1, uxy_l1; uxy_l1, uyy_l1];

Omega = [0 2; -2 0];

A2D = [zeros(2), eye(2); U_star_XX, Omega];

[V, D] = eig(A2D);

oscillatory_evec = real(V(:,3));

oscillatory_pos_mag = norm([oscillatory_evec(1), oscillatory_evec(2)]);

pos_mag_req = 0.0001;

oscillatory_mag_factor = pos_mag_req / oscillatory_pos_mag;

oscillatory_ic = oscillatory_evec .* oscillatory_mag_factor;

% Time is one period
t = linspace(0, 2*pi/imag(oscillatory_eval), 1000);

xi_0 = oscillatory_ic(1);
xi_dot_0 = oscillatory_ic(3);
eta_0 = oscillatory_ic(2);
eta_dot_0 = oscillatory_ic(4);
x0 = [l1_pos(1) + xi_0; l1_pos(2) + eta_0; 0; xi_dot_0; eta_dot_0; 0];
```

Problem 1

```
% Set tolerance for numerical integrator
TOL = 5e-14;

% Set options for ode113
options = odeset('RelTol', TOL, 'AbsTol', TOL);

% Arbitrary initial state for demonstration
state0 = [1, 0, 0, 1, 0, 0]; % [m, m/s]
phi0 = eye(6); % STM matrix evaluated at initial condition is identity
phi0_row = reshape(phi0, [6, 6]);

state_phi_0 = [state0; phi0_row];
```

```
[t_out, state_out] = ode113(@(t,state)CR3BP_full(state, mu), [0, 1],
state_phi_0, options);
```

Problem 2

```
% Set options for ode113
options = odeset('RelTol', TOL, 'AbsTol', TOL);
[tout, xout] = ode113(@(t, state)CR3BP(state, mu), [0 t(end)], x0, options);

V0 = [x0; t(end)];

V_soln = gen_3d_periodic_orbit_single_shooting(V0, mu, true);

[tout_corrected, xout_corrected] = ode113(@(t, state)CR3BP(state, mu), [0,
V_soln(end)], V_soln(1:6), options);

figure()
scatter(l1_pos(1), l1_pos(2), 'filled', 'black')
hold on
scatter(V_soln(1), V_soln(2), 'filled', 'blue')
plot(xout_corrected(:,1), xout_corrected(:,2), 'LineWidth',2)
hold off
axis equal
legend("L1", "Initial State", "Corrected Trajectory")
grid on
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
hold off
title("L1 Lyapunov Orbit - Corrected Trajectory")

figure()
plot(xout(:,1), xout(:,2), 'LineWidth',2)
hold on
scatter(l1_pos(1), l1_pos(2), 'filled', 'black')
scatter(xi_0 + l1_pos(1), eta_0 + l1_pos(2), 'filled', 'blue')

plot(xout_corrected(:,1), xout_corrected(:,2), 'LineWidth',2)
axis equal
legend("CR3BP Propagation", "L1", "Initial State", "Corrected Trajectory")
grid on
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
hold off
title("L1 Lyapunov Orbit - Initial and Corrected Trajectory")
```

Problem 3

```
V_family = natural_param_continuation(V_soln, mu);


% Plot all family members
```

```
p1_pos = [-mu, 0, 0];
p2_pos = [1-mu, 0, 0];

figure()
scatter(l1_pos(1), l1_pos(2), 'filled', 'red')
hold on
scatter(p1_pos(1), p1_pos(2), 'filled', 'blue')
scatter(p2_pos(1), p2_pos(2), 'filled', 'black')

for i = 1:10:size(V_family, 2)
    [tout, xout] = ode113(@(t,state)CR3BP(state, mu), [0, V_family(7,i)],
V_family(1:6,i), options);
    plot(xout(:,1), xout(:,2), 'LineWidth',2)
end
hold off
legend("L1", "Earth", "Moon")
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
grid on
axis equal
title("L1 Lyapunov Orbit Family")

figure()
for i = 1:size(V_family,2)
    x = V_family(1,i);
    y = V_family(2,i);
    z = V_family(3,i);
    x_dot = V_family(4,i);
    y_dot = V_family(5,i);
    z_dot = V_family(6,i);
    r1 = sqrt((x+mu)^2 + y^2 + z^2);
    r2 = sqrt((x-1+mu)^2 + y^2 + z^2);
    C(i) = x^2 + y^2 + z^2 + 2*(1-mu)/r1 + mu/r2 - (x_dot^2 + y_dot^2 +
z_dot^2);
end
plot(V_family(end,:)*t_star_em/86400, C, 'o')
hold on
scatter(V_family(end,1)*t_star_em/86400, C(1), 'filled', 'red')
scatter(V_family(end,end)*t_star_em/86400, C(end), 'filled', 'black')
legend("Jacobi Constant", "Start", "End")
xlabel("Period [days]")
ylabel("Jacobi Constant [-]")
grid on
title("Jacobi Constant and Period along the Family")
```

Problem 4

```
% Given
x0 = [0.82340, 0, -0.026755,0,0.13742,0]';
T = 2.7477;
V0 = [x0; T];

% Set options for ode113
```

```
options = odeset('RelTol', 5e-14, 'AbsTol', 5e-14);

V_soln = gen_3d_periodic_orbit_single_shooting(V0, mu, true);

[tout_corrected, xout_corrected] = ode113(@(t, state)CR3BP(state, mu), [0,
V_soln(end)], V_soln(1:6), options);

% Earth Moon system equilibrium points
[em_eq_pts, em_eq_validity] = all_eq_points(mu);

% Only looking at L1 eq point planar oscillatory modes
l1_pos = [em_eq_pts(1,:), 0];

% Set options for ode113
[tout, xout] = ode113(@(t, state)CR3BP(state, mu), [0 T], x0, options);

p1_pos = [-mu, 0, 0];
p2_pos = [1-mu, 0, 0];

figure()
hold on
scatter3(l1_pos(1), l1_pos(2), l1_pos(3), 'filled', 'red')
hold on
scatter3(x0(1), x0(2), x0(3), 'filled', 'blue')
plot3(xout_corrected(:,1), xout_corrected(:,2), xout_corrected(:,3),
'LineWidth',2)
% scatter3(p1_pos(1), p1_pos(2), p1_pos(3), 'filled', 'blue')
% scatter3(p2_pos(1), p2_pos(2), p2_pos(3), 'filled', 'black')
hold off
legend("L1", "Provided Initial Guess", "Corrected Trajectory")
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
zlabel('$$\hat{z}$$','Interpreter','Latex', 'FontSize',18)
axis equal
grid on
title("L1 Corrected Halo Orbit w/o the Moon")

figure()
plot3(xout(:,1), xout(:,2), xout(:,3), 'LineWidth',2)
hold on
scatter3(l1_pos(1), l1_pos(2), l1_pos(3), 'filled', 'red')
scatter3(x0(1), x0(2), x0(3), 'filled', 'blue')
plot3(xout_corrected(:,1), xout_corrected(:,2), xout_corrected(:,3),
'LineWidth',2)
% scatter3(p1_pos(1), p1_pos(2), p1_pos(3), 'filled', 'blue')
scatter3(p2_pos(1), p2_pos(2), p2_pos(3), 'filled', 'black')
hold off
legend("CR3BP Propagation", "L1", "Initial Guess", "Corrected Trajectory")
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
zlabel('$$\hat{z}$$','Interpreter','Latex', 'FontSize',18)
axis equal
grid on
title("L1 Halo Orbit corrected trajectory")
```

Problem 5

```
V_family = pseudo_arc_length_continuation(V_soln, mu);

% Plot all family members

p1_pos = [-mu, 0, 0];
p2_pos = [1-mu, 0, 0];

figure()
for i = 1:size(V_family,2)
    x = V_family(1,i);
    y = V_family(2,i);
    z = V_family(3,i);
    x_dot = V_family(4,i);
    y_dot = V_family(5,i);
    z_dot = V_family(6,i);
    r1 = sqrt((x+mu)^2 + y^2 + z^2);
    r2 = sqrt((x-1+mu)^2 + y^2 + z^2);
    C(i) = x^2 + y^2 + z^2 + 2*(1-mu)/r1 + mu/r2 - (x_dot^2 + y_dot^2 +
z_dot^2);
end
plot(V_family(end,:)*t_star_em/86400, C, 'o')
hold on
scatter(V_family(end,1)*t_star_em/86400, C(1), 'filled', 'red')
scatter(V_family(end,end)*t_star_em/86400, C(end), 'filled', 'black')
legend("Jacobi Constant", "Start", "End")
xlabel("Period [days]")
ylabel("Jacobi Constant [-]")
grid on
title("Jacobi Constant and Period along the Family")

figure()
subplot(1,2,1)
scatter3(l1_pos(1), l1_pos(2), l1_pos(3), 'filled', 'red')
hold on
scatter3(p2_pos(1), p2_pos(2), p2_pos(3), 'filled', 'black')

for i = 1:100:size(V_family, 2)
    [tout, xout] = ode113(@(t,state)CR3BP(state, mu), [0, V_family(7,i)],
V_family(1:6,i), options);
    plot3(xout(:,1), xout(:,2), xout(:,3), 'LineWidth',2)
end
hold off
legend("L1", "Moon")
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
title("L1 Halo Orbit Family")
axis equal

subplot(1,2,2)
scatter3(l1_pos(1), l1_pos(2), l1_pos(3), 'filled', 'red')
hold on
```

```
scatter3(p2_pos(1), p2_pos(2), p2_pos(3), 'filled', 'black')

for i = 1:100:size(V_family, 2)
    [tout, xout] = ode113(@(t,state)CR3BP(state, mu), [0, V_family(7,i)],
V_family(1:6,i), options);
    plot3(xout(:,1), xout(:,2), xout(:,3), 'LineWidth',2)
end
hold off
legend("L1", "Moon")
xlabel('$$\hat{x}$$','Interpreter','Latex', 'FontSize',18)
ylabel('$$\hat{y}$$','Interpreter','Latex', 'FontSize',18)
zlabel('$$\hat{z}$$','Interpreter','Latex', 'FontSize',18)
title("L1 Halo Orbit Family")
axis equal
```

Functions

```
function out = u_xx(mu, x_eq)
    % Pseudo potential function partial derivative wrt x, x at eq point
    % Assuming z = 0
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = 1 - (1-mu)/(r1^3) - mu/(r2^3) + (3*(1-mu)*(x+mu)^2)/(r1^5) +
(3*mu*(x-1+mu)^2)/(r2^5);
end

function out = u_xy(mu, x_eq)
    % Pseudo potential function partial derivative wrt x, y at eq point
    % Assuming z = 0
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = (3*(1-mu)*(x+mu)*y)/(r1^5) + (3*mu*y*(x-1+mu))/(r2^5);
end

function out = u_xz(mu, x_eq)
    % Pseudo potential function partial derivative wrt x, z at eq point
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = 3*(1-mu)*(x+mu)*z/r1^5 + 3*mu*(x-1+mu)*z/r2^5;
end

function out = u_yy(mu, x_eq)
    % Pseudo potential function partial derivative wrt y, y at eq point
    % Assuming z = 0
```

```
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = 1 - (1-mu)/(r1^3) - mu/(r2^3) + (3*(1-mu)*y^2)/(r1^5) + (3*mu*y^2)/
(r2^5);
end

function out = u_yz(mu, x_eq)
    % Pseudo potential function partial derivative wrt y, z at eq point
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = 3*(1-mu)*y*z/r1^5 + 3*mu*y*z/r2^5;
end

function out = u_zz(mu, x_eq)
    % Pseudo potential function partial derivative wrt z, z at eq point
    % Assuming z = 0
    x = x_eq(1);
    y = x_eq(2);
    z = x_eq(3);
    r1 = sqrt((x + mu)^2 + y^2 + z^2);
    r2 = sqrt((x - 1 + mu)^2 + y^2 + z^2);
    out = -(1-mu)/(r1^3) - mu/(r2^3) + (3 * (1-mu) * z^2)/(r1^5) + (3*mu*z^2)/
(r2^5);
end

function V_family = pseudo_arc_length_continuation(V_soln, mu)
    % V_soln - First corrected free variable solution
    % mu - system parameter

    % out - Family of free variables

    a = 384400; % [km] Average earth-moon semi-major axis
    r_E = 6378.1363; % [km] Earth equatorial radius (Vallado, Appendix D)
    r_Moon = 1738; % [km] Moon equatorial radius (Vallado, Appendix D)
    r_E_normalized = r_E/a;
    r_Moon_normalized = r_Moon/a;

    TOL = 1e-12;

    V_family = V_soln;
    delta_s = 1e-3;

    % Set options for ode113
    options = odeset('RelTol', TOL, 'AbsTol', TOL);

    [tout_corrected, xout_corrected] = ode113(@(t, state)CR3BP(state, mu),
[0, V_soln(end)], V_soln(1:6), options);
```

```
    % Check if initial position is outside the P1, P2 bodies' normalized
radius
    initial_x_pos = V_family(1:3,1);
    p1_pos = [-mu, 0, 0]';
    p2_pos = [1-mu, 0, 0]';
    p1_minus_init_pos = p1_pos - initial_x_pos;
    p2_minus_init_pos = p2_pos - initial_x_pos;
    p1_or_p2_bound = false;
    if (norm(p1_minus_init_pos) <= r_E_normalized)
        p1_or_p2_bound = true;
    end
    if (norm(p2_minus_init_pos) <= r_E_normalized)
        p1_or_p2_bound = true;
    end

    % Initialize nHat
    phi0 = reshape(eye(6), [36,1]);
    state0 = [V_soln(1:6); phi0];
    [tout, state_out] = ode113(@(t, state)CR3BP_full(state, mu), [0,
V_soln(end)], state0, options);
    statef = state_out(end, :)';
    state_dot = CR3BP_full(statef, mu);
    DF_d = DF_mat(statef, state_dot);
    prev_n_hat = null(DF_d);

    family_member = 1;
    % max_family_members = 2000;
    max_family_members = 20; % ONLY FOR APPENDIX CODE

    % for i = 1:family_members
    while ((~p1_or_p2_bound) && (family_member <= max_family_members))
        print_out = sprintf('Family member - %d', family_member);
        disp(print_out)
        Vd = V_family(:,family_member);
        state0 = [Vd(1:6); phi0];
        [tout, state_out] = ode113(@(t, state)CR3BP_full(state, mu), [0,
Vd(end)], state0, options);
        statef = state_out(end, :)';
        state_dot = CR3BP_full(statef, mu);
        DF_d = DF_mat(statef, state_dot);
        n_hat = null(DF_d);

        if dot(n_hat, prev_n_hat) < 0
            n_hat = -1 * n_hat;
        end

        n_hat = n_hat/norm(n_hat);

        V_star = V_family(:,family_member);
        V = V_star + delta_s*n_hat;
        H_norm = norm(H_psal(V_family(:,family_member), V_star, delta_s,
n_hat, statef));
        counter = 1;
```

```
        % Check if initial position is outside the P1, P2 bodies' normalized
radius
        % [tout, xout] = ode113(@(t, state)CR3BP(state, mu), [0, V(end)],
V(1:6), options);
        % for i = 1:length(tout)
        %     p2_minus_pos = p2_pos' - xout(i,1:3);
        %     if (norm(p2_minus_pos) < r_Moon_normalized)
        %         p1_or_p2_bound = true;
        %     end
        %     p1_minus_pos = p1_pos' - xout(i,1:3);
        %     if (norm(p1_minus_pos) < r_E_normalized)
        %         p1_or_p2_bound = true;
        %     end
        % end
        initial_x_pos = V_family(1:3,family_member);
        p1_minus_init_pos = p1_pos - initial_x_pos;
        p2_minus_init_pos = p2_pos - initial_x_pos;
        p1_or_p2_bound = false;
        if (norm(p1_minus_init_pos) <= r_E_normalized)
            p1_or_p2_bound = true;
        end
        if (norm(p2_minus_init_pos) <= r_E_normalized)
            p1_or_p2_bound = true;
        end

        while ((H_norm > TOL) && (counter < 20))
            V_i = V(:,counter);
            state0 = [V_i(1:6); phi0];

            [tout, state_out] = ode113(@(t, state)CR3BP_full(state, mu), [0,
V_i(end)], state0, options);

            statef = state_out(end, :)';
            state_dot = CR3BP_full(statef, mu);
            H_V = H_psal(V_i, V_star, delta_s, n_hat, statef);
            H_norm = norm(H_V);
            DH_V = DH_mat_psal(statef, state_dot, n_hat);
            V_ip1 = V_i - inv(DH_V) * H_V;
            V(:,counter+1) = V_ip1;
            counter = counter + 1;
        end
        prev_n_hat = n_hat;

        V_family(:,family_member+1) = V(:,end);
        family_member = family_member + 1;
    end

end

function V_family = natural_param_continuation(V_soln, mu)
    % V_soln - First corrected free variable solution
    % mu - system parameter

    % out - Family of free variables
```

```
    % ASSUMPTION - Earth-Moon System!

    a = 384400; % [km] Average earth-moon semi-major axis
    r_E = 6378.1363; % [km] Earth equatorial radius (Vallado, Appendix D)
    r_Moon = 1738; % [km] Moon equatorial radius (Vallado, Appendix D)
    r_E_normalized = r_E/a;
    r_Moon_normalized = r_Moon/a;

    TOL = 5e-14;

    V_family = V_soln;
    delta = -2e-3;

    % Set options for ode113
    options = odeset('RelTol', TOL, 'AbsTol', TOL);

    [tout_corrected, xout_corrected] = ode113(@(t, state)CR3BP(state, mu),
[0, V_soln(end)], V_soln(1:6), options);

    % Check if initial position is outside the P1, P2 bodies' normalized
radius
    initial_x_pos = V_family(1:3,1);
    p1_pos = [-mu, 0, 0]';
    p2_pos = [1-mu, 0, 0]';
    p1_minus_init_pos = p1_pos - initial_x_pos;
    p2_minus_init_pos = p2_pos - initial_x_pos;
    p1_or_p2_bound = false;
    if (norm(p1_minus_init_pos) <= r_E_normalized)
        p1_or_p2_bound = true;
    end
    if (norm(p2_minus_init_pos) <= r_E_normalized)
        p1_or_p2_bound = true;
    end

    family_member = 1;
    % max_family_members = 150;
    max_family_members = 15; %% ONLY FOR APPENDIX CODE

    while ((~p1_or_p2_bound) && (family_member <= max_family_members))
        print_out = sprintf('Family member - %d', family_member);
        disp(print_out)
        V = V_family(:,family_member);
        x_star = V(1);
        x_star_plus_delta = x_star + delta;
        H_norm = norm(H(V_soln(1:6), xout_corrected(end,:),
x_star_plus_delta));
        counter = 1;

        % Check if initial position is outside the P1, P2 bodies' normalized
radius
        initial_x_pos = V_family(1:3,family_member);
        p1_minus_init_pos = p1_pos - initial_x_pos;
        p2_minus_init_pos = p2_pos - initial_x_pos;
        p1_or_p2_bound = false;
```

```
        if (norm(p1_minus_init_pos) <= r_E_normalized)
            p1_or_p2_bound = true;
        end
        if (norm(p2_minus_init_pos) <= r_E_normalized)
            p1_or_p2_bound = true;
        end

        while ((H_norm > TOL) && (counter < 10))
            V_i = V(:,counter);
            phi_0 = reshape(eye(6), [36,1]);
            state0 = [V_i(1:6); phi_0];

            [tout, state_out] = ode113(@(t, state)CR3BP_full(state, mu), [0,
V_i(end)], state0, options);

            statef = state_out(end, :)';
            state_dot = CR3BP_full(statef, mu);
            H_V = H(V_i(1:6), statef, x_star_plus_delta);
            H_norm = norm(H_V);
            DH_V = DH_mat(statef, state_dot);
            try
                inv(DH_V);
            catch
                break
            end
            V_ip1 = V_i - inv(DH_V) * H_V;
            V(:,counter+1) = V_ip1;
            counter = counter + 1;
        end

        V_family(:,family_member+1) = V(:,end);
        family_member = family_member + 1;
    end
end

function out = in_plane_modes(mu, x_eq)
    % Calculate four in plane modes for eq points
    uxx = u_xx(mu, x_eq);
    uyy = u_yy(mu, x_eq);
    uzz = u_zz(mu, x_eq);
    uxy = u_xy(mu, x_eq);
    Lambda_1 = (-4+uxx+uyy)/2 + (sqrt((4-uxx-uyy)^2 - 4*(uxx*uyy - uxy^2)))/2;
    Lambda_2 = (-4+uxx+uyy)/2 - (sqrt((4-uxx-uyy)^2 - 4*(uxx*uyy - uxy^2)))/2;
    lambda_1 = sqrt(Lambda_1);
    lambda_2 = -sqrt(Lambda_1);
    lambda_3 = sqrt(Lambda_2);
    lambda_4 = -sqrt(Lambda_2);
    out = [lambda_1, lambda_2, lambda_3, lambda_4];
end

function out = H(state0, statef, x_star_plus_delta)
    out = [statef(1) - state0(1);
            statef(2) - state0(2);
            statef(3) - state0(3);
```

```
            statef(4) - state0(4);
            statef(6) - state0(6);
            state0(2);
            state0(1) - x_star_plus_delta];
end

function out = H_psal(V, V_star, delta_s, n_hat, statef)
    out = [statef(1) - V(1);
            statef(2) - V(2);
            statef(3) - V(3);
            statef(4) - V(4);
            statef(6) - V(6);
            V(2);
            dot(V-V_star, n_hat) - delta_s];
end

function V_soln = gen_3d_periodic_orbit_single_shooting(V0, system_params,
plot_input)
    % Script to compute a general three-dimensional periodic orbit via single
shooting
    % Inputs
    % V0 - initial guess for a free variable vector
    % statef_V0 - final state when V0 is used as initial guess using CR3BRP
    % EOMs
    % system_params - system parameters
    %
    % Output
    % V_soln - free variable vector corresponding to a solution

    % Get mass ratio of system
    mu = system_params(1);

    % Set tolerance for numerical integrator and constraint vector
    TOL = 5e-14;

    % Set options for ode113
    options = odeset('RelTol', TOL, 'AbsTol', TOL);

    % Propagate V0 non-linear CR3BP EOMs
    [tout, xout] = ode113(@(t, state)CR3BP(state, mu), [0 V0(end)], V0(1:6),
options);

    % Final final variables using V0
    statef_V0 = xout(end,:);

    % Period is a free variable
    T = V0(end);

    % Initialize constraint vector norm
    F_norm(1) = norm(F(V0(:,1), statef_V0));

    % Matrix of all free variable vectors
    V(:,1) = V0;
```

```
    % While loop params
    counter = 1;
    counter_max = 50;

    % While loop to reduce F_norm
    while ((F_norm(counter) > TOL) && (counter < counter_max))
        phi0 = reshape(eye(6), [36, 1]); % Initial phi is identity
        state0 = [V(1:6,counter); phi0];

        % Propagate full state and STM
        [t_out, state_out] = ode113(@(t,state)CR3BP_full(state, mu), [0,
V(7,counter)], state0, options);
        statef = state_out(end, :)';
        state_dot = CR3BP_full(statef, mu);
        F_i = F(state0, statef);
        DF_i = DF_mat(statef, state_dot);

        % Find V_i+1
        V(:,counter+1) = V(:,counter) - DF_i' * inv(DF_i * DF_i') * F_i;

        % Calculate F_norm and update counter
        F_norm(counter+1) = norm(F_i);
        counter = counter + 1;
    end

    if plot_input
        figure()
        plot([1:counter], F_norm, '-o', 'LineWidth', 2)
        yscale log
        grid on
        xlabel("Iterations")
        ylabel("F Norm")
        title("Constraint Vector Norm for each Iteration")
        hold on
        tol_yline = ones([counter,1])*TOL;

        plot([1:counter], tol_yline, 'red', 'LineWidth', 2)
        hold off
        legend("Norm", "Threshold")
    end

    print_out = sprintf('Difference in y_dot_0^2 and y_dot_f^2 - %d', V0(5)^2
- V(5,end)^2);
    disp(print_out)

    V_soln = V(:,end);

end

function out = F(state0, statef)
    % Modified Constraint Vector
    out = [statef(1) - state0(1); % x0
            statef(2) - state0(2); % y0
            statef(3) - state0(3); % z0
```

```
            statef(4) - state0(4); % x0_dot
            statef(6) - state0(6); % z0_dot
            state0(2)]; % y0
end

function out = DH_mat_psal(state, state_dot, n_hat)
    phi_row = state(7:end);
    phi_mat = reshape(phi_row, [6,6])';
    phi_minus_I = phi_mat - eye(6);
    phi_mod = [phi_minus_I(1:4, :); phi_minus_I(6,:)];

    state_dot_col = [state_dot(1:4); state_dot(6)];

    out = [phi_mod, state_dot_col; 0 1 0 0 0 0 0; n_hat'];
end

function out = DH_mat(state, state_dot)
    phi_row = state(7:end);
    phi_mat = reshape(phi_row, [6,6])';
    phi_minus_I = phi_mat - eye(6);
    phi_mod = [phi_minus_I(1:4, :); phi_minus_I(6,:)];

    state_dot_col = [state_dot(1:4); state_dot(6)];

    out = [phi_mod, state_dot_col; 0 1 0 0 0 0 0; 1 0 0 0 0 0 0];
end

function out = DF_mat(state, state_dot)
    % Modified constraint DF matrix

    % Convert STM to matrix from row vector
    phi_row = state(7:end);
    phi_mat = reshape(phi_row, [6,6])';
    % Subtract identity and extract all rows except 5th row
    phi_minus_I = phi_mat - eye(6);
    phi_mod = [phi_minus_I(1:4, :); phi_minus_I(6,:)];

    % Grab states - x_dot, ydot, zdot, xdotdot, zdotdot
    state_dot_col = [state_dot(1:4); state_dot(6)];

    out = [phi_mod, state_dot_col; 0 1 0 0 0 0 0];
end

function state_dot = CR3BP(state, mu)
    % Circular Restricted 3 Body Problem non-dimensional EOMs
    x = state(1);
    y = state(2);
    z = state(3);
    xdot = state(4);
    ydot = state(5);
    zdot = state(6);

    r1 = sqrt((x + mu)^2 + (y)^2 + (z)^2);
    r2 = sqrt((x - 1 + mu)^2 + (y)^2 + (z)^2);
```

```
    state_dot(1, 1) = xdot;
    state_dot(2, 1) = ydot;
    state_dot(3, 1) = zdot;

    state_dot(4, 1) = 2*ydot + x - (1 - mu)*(x + mu)/(r1^3) - mu * (x - 1 +
mu)/(r2^3);
    state_dot(5, 1) = -2*xdot + y - (1 - mu)*y/(r1^3) - mu*y/(r2^3);
    state_dot(6, 1) = - (1 - mu)*z/(r1^3) - mu*z/(r2^3);
end

function state_phi_dot = CR3BP_full(state_phi, mu)
    % Full state vector and state transition matrix differential equation
    % Inputs:
    % state_phi - Augmented state vector and STM [42x1]. The state vector -
    % [x0, y0, z0, x0_dot, y0_dot, z0_dot]. The STM - is 6x6 with each
    % element described as - phi_ij = dxi(tf)/dxj(t0). The phi matrix is
    % reshaped such that all the rows are concatenated vertically. For
    % example -
    % phi_mat = [phi11, phi12, phi13, ..., phi16;
    %            [phi21, phi22, phi23, ..., phi26;
    %            ...
    %            [phi61, phi62, phi63, ..., phi66]
    % becomes
    % phi_row = [phi11, phi12, ..., phi16, phi21, phi22, ..., phi66]'
    %
    % mu - system mass ratio [-]
    %
    % Output
    % state_phi_dot - Augmented state vector dot and STM_dot [42x1]. The
    % augmentation and reshaping scheme remains the same as the input.

    x = state_phi(1);
    y = state_phi(2);
    z = state_phi(3);
    xdot = state_phi(4);
    ydot = state_phi(5);
    zdot = state_phi(6);

    r1 = sqrt((x + mu)^2 + (y)^2 + (z)^2);
    r2 = sqrt((x - 1 + mu)^2 + (y)^2 + (z)^2);

    state_dot(1, 1) = xdot;
    state_dot(2, 1) = ydot;
    state_dot(3, 1) = zdot;

    state_dot(4, 1) = 2*ydot + x - (1 - mu)*(x + mu)/(r1^3) - mu * (x - 1 +
mu)/(r2^3);
    state_dot(5, 1) = -2*xdot + y - (1 - mu)*y/(r1^3) - mu*y/(r2^3);
    state_dot(6, 1) = -(1 - mu)*z/(r1^3) - mu*z/(r2^3);

    % Calc pseudo-potentials
    uxx = u_xx(mu, [x, y, z]);
    uyy = u_yy(mu, [x, y, z]);
```

```
    uxy = u_xy(mu, [x, y, z]);
    uzz = u_zz(mu, [x, y, z]);
    U_mat = [uxx, uxy 0; uxy, uyy 0; 0 0 uzz];
    Omega = [0 2 0; -2 0 0; 0 0 0];
    A = [zeros(3), eye(3);
        U_mat, Omega];

    % Get only the phi elements into a row
    phi_row = state_phi(7:end);

    % Converting phi to matrix
    phi_mat = reshape(phi_row, [6,6])';

    % Get phi_dot
    phi_dot_mat = A * phi_mat;

    % Convert back to row
    phi_dot_row = reshape(phi_dot_mat', [36,1]);

    % Augment state and phi (in row form)
    state_phi_dot = [state_dot; phi_dot_row];

end
```

Published with MATLAB® R2024a