MAE 5419 PROJECT

Subtitle – Low Thrust LEO to GEO Problem

Abstract

The main of purpose of this project is to minimize fuel consumption for a satellite going from LEO to GEO. This problem can be compared to a two-point boundary problem with a given initial value and final values. Therefore, the best approach is to use calculus of variations which uses admissible controls and trajectories to find the optimal control to get an optimal trajectory to get to the desired final states. Even though the algorithm does not work as planned, there was a lot to learn from whilst working on this project. By writing out each line of code and troubleshooting, more of an understanding of how the variation of extremals process works was established. The source of error in the algorithm was within solving for the influence function which led to the problem not converging.

By: Devin Quach

Contents

Introduction:

Background:

Satellites are generally orbiting in the geostationary orbit (GEO) around the Earth. This orbit is a circular, equatorial, and geosynchronous orbit 42,157 km in radius from the Earth's center which follows the Earth's rotation. The fact that it follows Earth's rotation makes it a great orbit for communication, meteorology, and navigation satellites. The low Earth orbit (LEO) is a circular orbit, closer to Earth with a range of 6,531-8,371 km in radius from Earth's center. This orbit is mainly used for imaging applications, military reconnaissance, and communications. Also, since this problem will be dealing with two bodies the standard gravitational parameter of Earth, mu=3.98e^5 km^3 / s^2 .

Big Picture/Purpose:

The main of purpose of this project is to minimize fuel consumption for a satellite going from LEO to GEO. The initial state is specified by the properties of a LEO orbit and the final state and time are specified by the properties of a GEO orbit and an assumed time of flight.

Problem Approach:

This problem can be compared to a two-point boundary problem with a given initial value and final values. Therefore, the best approach is to use calculus of variations which uses admissible controls and trajectories to find the optimal control to get an optimal trajectory to get to the desired final states. The state vector will consist of [Radial Distance, Radial Velocity, Tangential Velocity], therefore, the initial state will consist of LEO properties and the final state will consist of GEO properties.

$$\overline{\dot{X}_{initial}} = [6{,}531km \quad 0\frac{rad}{second} \quad \sqrt{\frac{mu}{6{,}531}}\frac{km}{s}] \tag{1}$$

$$\overline{\dot{X}_{final}} = [42{,}157km \quad 0\frac{rad}{second} \quad \sqrt{\frac{mu}{42{,}157}}\frac{km}{s}] \tag{2}$$

Since the goal is to minimize fuel, it is also to minimize thrust, therefore, the objective function (J) is given as:

$$J = \int |T| dt \tag{3}$$

Where T is the thrust and t is time. Then the Hamiltonian with respective necessary conditions is used to find the equations to solve for the optimal control. To numerically solve an optimization problem like this two methods can be used: Steepest Descent and Variation of Extremals.

Methods:

There are two algorithms that could be used to solve this problem numerically: steepest descent and variation of extremals.

Theory Developed/Process:

The method that was chosen was the variation of extremals method. This was mainly due to more comfortability and experience with the variation of extremals method compared to the steepest descent method. The variation of extremals method uses initial states, final states, a guessed initial costate and a given final time to solve for the optimal controls. The process includes deriving the reduced differential equations and using the guessed initial costates to perturb the equation of motion and Pdot values. It then uses those perturb values to perturb new state and costate vectors using forward integration. Once the whole state and costate vectors have been fully perturbed till the final time, an influence function has to be created using finite difference. Generally, there are two influence functions Px and Pp, however, since the final state is specified the influence functions can be modified. The only influence function needed was Px and a new equation that gave the new initial costates.

Assumptions Used:

Assumptions had to be made to provide the satellite with physical properties and to define the change in orbit. These assumptions mostly apply to the equations of motion. The equations of motion were taken from Dr.Tragesser's example of a "transfer to max circular radius in given final time" in lecture 7. These equations can be seen in Appendix A.1. To use these equations the initial mass must be assumed and the change in mass must be constant. For context, figure 1 below is a diagram of the physical problem.

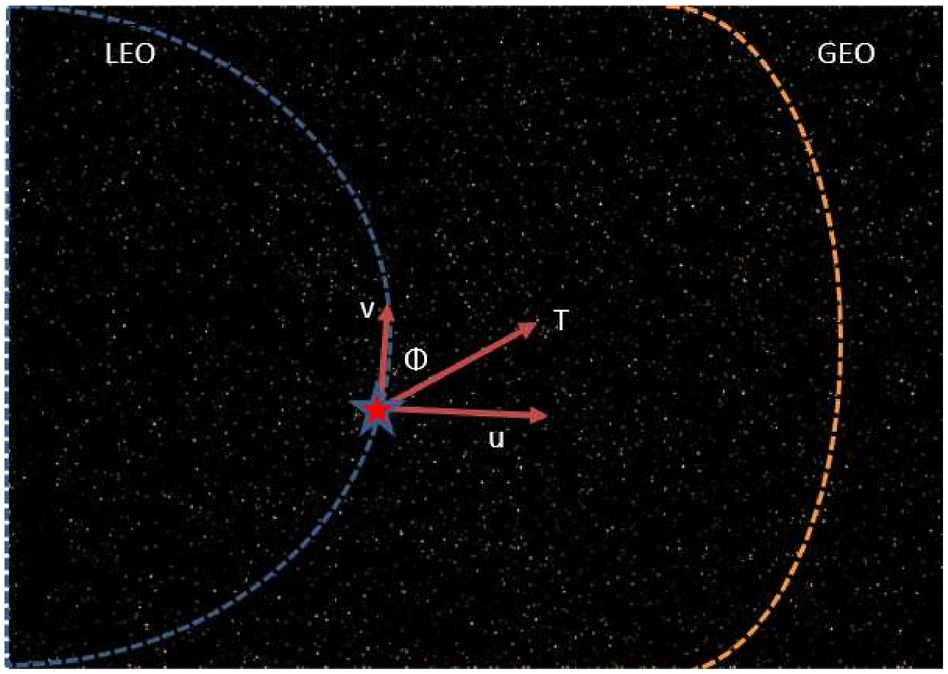


***Figure 1**: Diagram of Problem*

The derivation of the necessary equations using the Hamiltonian and necessary conditions can be seen in Appendix A.1. One thing to note in this derivation is that thrust is linear in H, therefore, when solving for dH/dT = 0 the equation was not useful as T was not in it. This then indicated the need to use Pontryagin's maximum principle, therefore, constraining the thrust capabilities to minimize the Hamiltonian. Now, instead of using the dH/DT = 0 necessary condition, $H(\overline{x^*}, \overline{u^*} + \delta\bar{u}, t) \leq H(\overline{x^*}, \overline{u^*}, t)$ was used to solve for thrust.

Numerical Results:

The current code does not accurately provide the optimal thrust and angle of thrust that minimizes the fuel. After iterating to get valid and clear results the code used these assumptions: initial mass = 3000 kg, change in mass = 0.1kg/s, time of flight 8.3 hours, highest thrust = 981/initial mass = 0.32 km/s^2, and the lowest thrust is 0 km/s^2. Going through the current variation of extremals algorithm provides the graphs below.

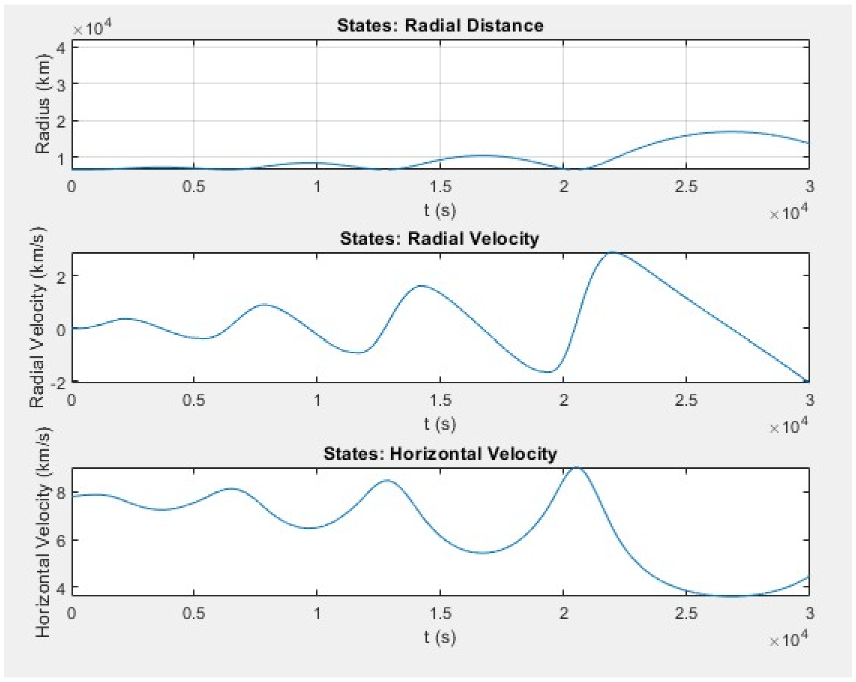


***Figure 2**: Optimal State Graphs with T_high is Dependent on Initial Mass*

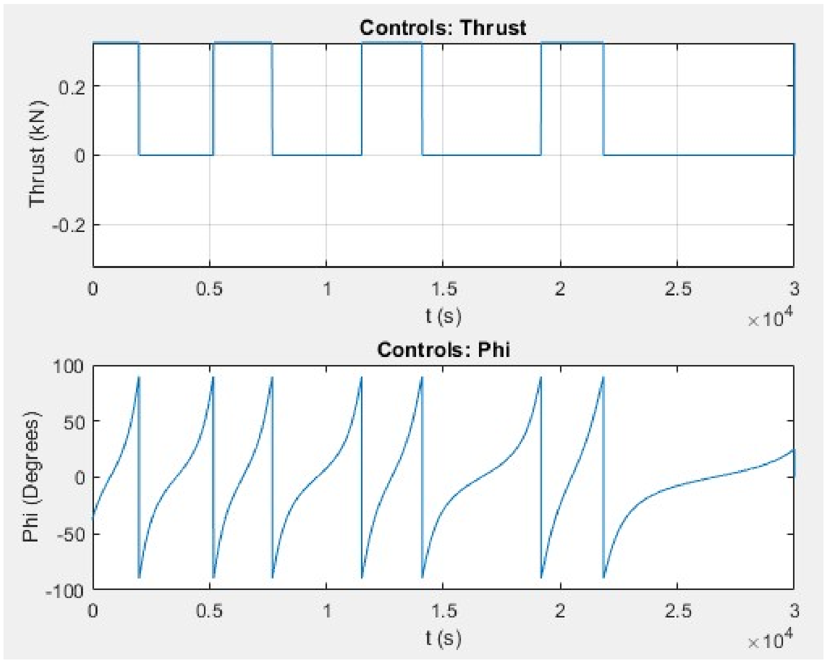


***Figure 3:** Optimal Control Graphs with T_high is Dependent on Initial Mass*

If the highest thrust is changed to be 1 km/s^2 then the resulting graphs are below.

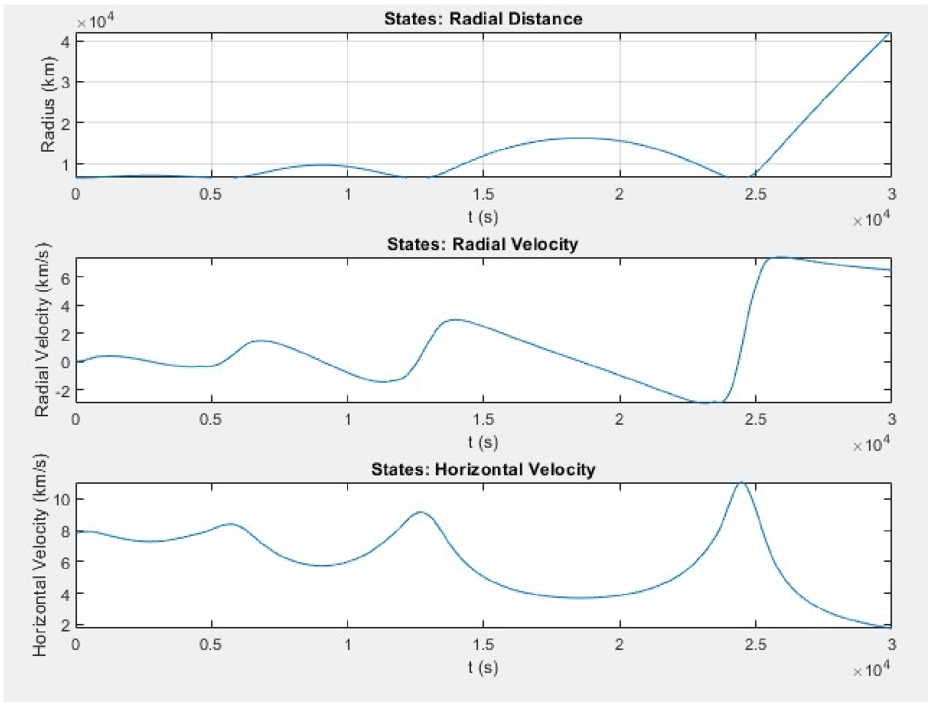


Figure 4: *Optimal State Graphs with T_high=1 km/s^2*

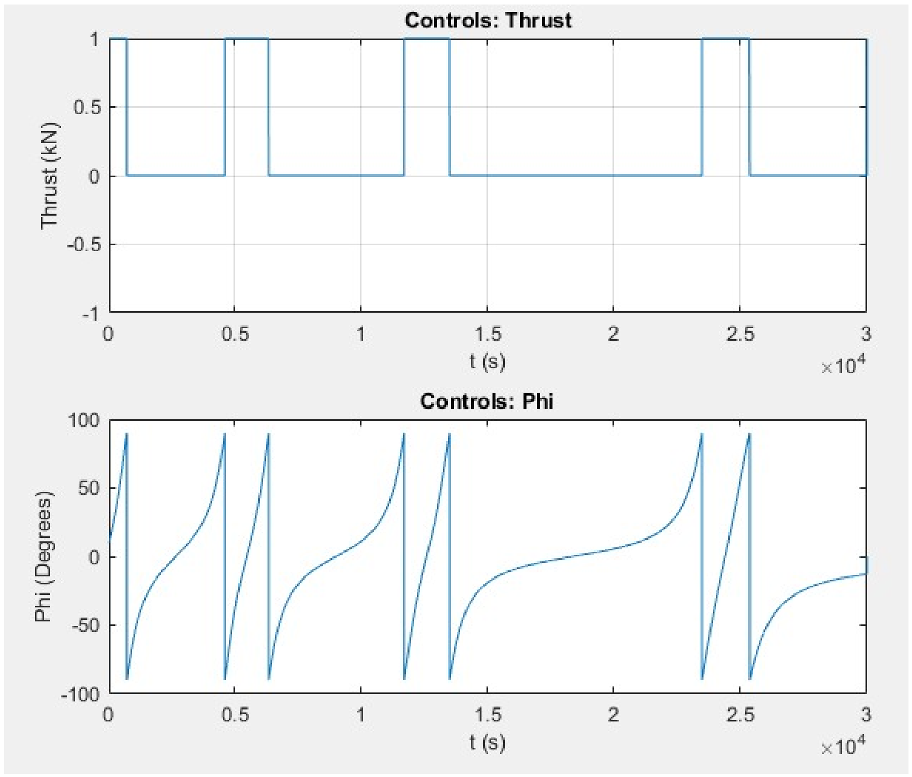


Figure 5*: Optimal Control with T_high = 1 km/s^2*

Conclusion:

Discussion:

The biggest issue with the code currently is the fact that the influence matrix Px becomes NaN through the second iteration, therefore, not allowing the controls to iterate through enough to converge and get a more accurate answer. Since the influence matrix is not working correctly and provides NaN's then it is as if the algorithm is not going through the influence function. This in turn prevents the end condition to actually execute. This why as seen on figure 2 and 4, the radius does not reach the final radius or completely overshoots it. Also, the radial and horizontal velocities are not where they are desired to be. Figure 2 is close to the desired radial and horizontal velocity and figure 4 is much higher than the desired final velocities, therefore, overshooting the final orbit's radius. As for the controls, in both figure 3 and 5 the thrust toggles from full to no thrust as the angle of thrust flips from 90 degrees to -90 degrees. When phi is 90 degrees the satellite is positioned to move outwards, when it is -90 degrees it is positioned to move inwards, and a if it is 0 degrees then the satellite is moving in orbit.

As shown in figures 3 and 4, when T_high is less than 1 km/s^2 then the satellite is not able to reach the final radius of GEO. This means the thrust is too low for the satellite to reach the necessary orbit in the given amount of time. As shown in figure 5 and 6, when T_high is equal to 1 km/s^2 then the satellite overshoots the final radius. As the satellite approaches the final radius the phi should have became negative with full thrust to slow the satellite down. However, this did not happen due to the issue with the influence matrix.

A lot of troubleshooting and debugging was done to get to the current state of the algorithm. As seen above, the change in mass had to be set to 0.1 kg/s to allow the algorithm to plot. This in turn means the model assumes the mass is fairly constant and not changing. However, if it were set to 0 then the algorithm would not have worked. There was definitely a correlation with the initial mass and the change in mass as well as the given time of flight. The initial had to be bigger than the product of the change in mass and TOF but not too bigger. Therefore, a time of flight of 30000s and initial mass of 3000 kg were workable values. This is an area that needs to be further developed and explored.

Also, the initial guess for the costates is a very important part to the code. Initially it was running with a guess of 10e8 but as that turned out to be a bad guess the bootstrap method was used to find a better guess. The bootstrap method uses the initial guess for the costate from a simpler version of the problem. In this situation, the simpler version would be Dr.Tragesser's example of a "transfer to max circular radius in given final time" in lecture 7. The initial costates from Dr.T's code was then used as the initial guess for the costates in this problem, therefore, providing the graphs above.

Conclusion:

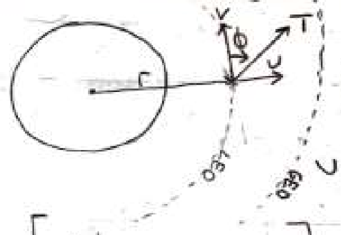
The main of purpose of this project is to minimize fuel consumption for a satellite going from LEO to GEO. Even though the algorithm does not work as planned, there was a lot to learn from whilst working on this project. By writing out each line of code and troubleshooting, more of an understanding of how the variation of extremals process works was established. The overall understanding of the variations of extremals that was stated above was flushed out and better understood.

The source of error in the algorithm was within solving for the influence function. This is also an area of improvement. If there was more time, a steepest descent method for solving the problem could be created and cross-referenced with this code to maybe solve the problem. All in all, this project was used to improve upon the understanding of calculus of variations, the variation of extremals method, debugging methods such as the bootstrap method, and to see a physical representation.

Appendix:

Appendix A.1: Handwritten Process for Finding Equations

LEO to GEO Project: $J = \int_{t_0}^{t_f} |T|\,dt$ ~ min fuel



$$\bar{x} = \begin{bmatrix} r \\ u \\ v \end{bmatrix} \qquad \bar{x}(t_0) = \begin{bmatrix} 6{,}531 - 8{,}371 \text{ km} \\ 0 \text{ km/s} \\ \sqrt{\mu/r_0} \text{ km/s} \end{bmatrix}$$

$$u = \begin{bmatrix} T \\ \phi \end{bmatrix} \qquad \bar{x}(t_f) = \begin{bmatrix} 42157 \text{ km} \\ 0 \text{ km/s} \\ \sqrt{\mu/r_f} \text{ km/s} \end{bmatrix}$$

$$\dot{\bar{x}} = \bar{a} = \begin{bmatrix} u \\ \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T\sin\phi}{m_0 - \dot{m}t} \\ -\frac{vu}{r} + \frac{T\cos\phi}{m_0 - \dot{m}t} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{x_3^2}{x_1} - \frac{\mu}{x_1^2} + \frac{T\sin\phi}{m_0 - \dot{m}t} \\ -\frac{x_3 x_2}{x_1} + \frac{T\cos\phi}{m_0 - \dot{m}t} \end{bmatrix}$$

$$H = g + p^T \bar{a}$$

$$= T + P_1 x_2 + P_2\left(\frac{x_3^2}{x_1} - \frac{\mu}{x_1^2} + \frac{T\sin\phi}{m_0 - \dot{m}t}\right) + P_3\left(-\frac{x_3 x_2}{x_1} + \frac{T\cos\phi}{m_0 - \dot{m}t}\right)$$

$$= T + P_1 x_2 + \frac{P_2 x_3^2}{x_1} - \frac{P_2 \mu}{x_1^2} + \frac{P_2 T\sin\phi}{m_0 - \dot{m}t} - \frac{P_3 x_3 x_2}{x_1} + \frac{P_3 T\cos\phi}{m_0 - \dot{m}t}$$

$$\dot{\bar{P}} = -\frac{\partial H}{\partial x} = -\begin{bmatrix} -\frac{P_2 x_3^2}{x_1^2} + \frac{2P_2\mu}{x_1^3} + \frac{P_3 x_3 x_2}{x_1^2} \\ P_1 - \frac{P_3 x_3}{x_1} \\ \frac{2P_2 x_3}{x_1} - \frac{P_3 x_2}{x_1} \end{bmatrix} = \begin{bmatrix} \frac{P_2 x_3^2}{x_1^2} - \frac{2P_2\mu}{x_1^3} - \frac{P_3 x_3 x_2}{x_1^2} \\ -P_1 + \frac{P_3 x_3}{x_1} \\ -\frac{2P_2 x_3}{x_1} + \frac{P_3 x_2}{x_1} \end{bmatrix}$$

$$\frac{\partial H}{\partial u} = \begin{bmatrix} 1 + \frac{P_2 \sin\phi}{m_0 - \dot{m}t} + \frac{P_3 \cos\phi}{m_0 - \dot{m}t} \\ + \frac{P_2 T\cos\phi}{m_0 - \dot{m}t} - \frac{P_3 T\sin\phi}{m_0 - \dot{m}t} \end{bmatrix} \overset{!}{=} \bar{0}$$

$$\frac{\partial H}{\partial \upsilon} = \left(\frac{T}{m_0 - \dot{m}t}\right)\left(-P_2\cos\phi - P_3\sin\phi\right) = 0$$

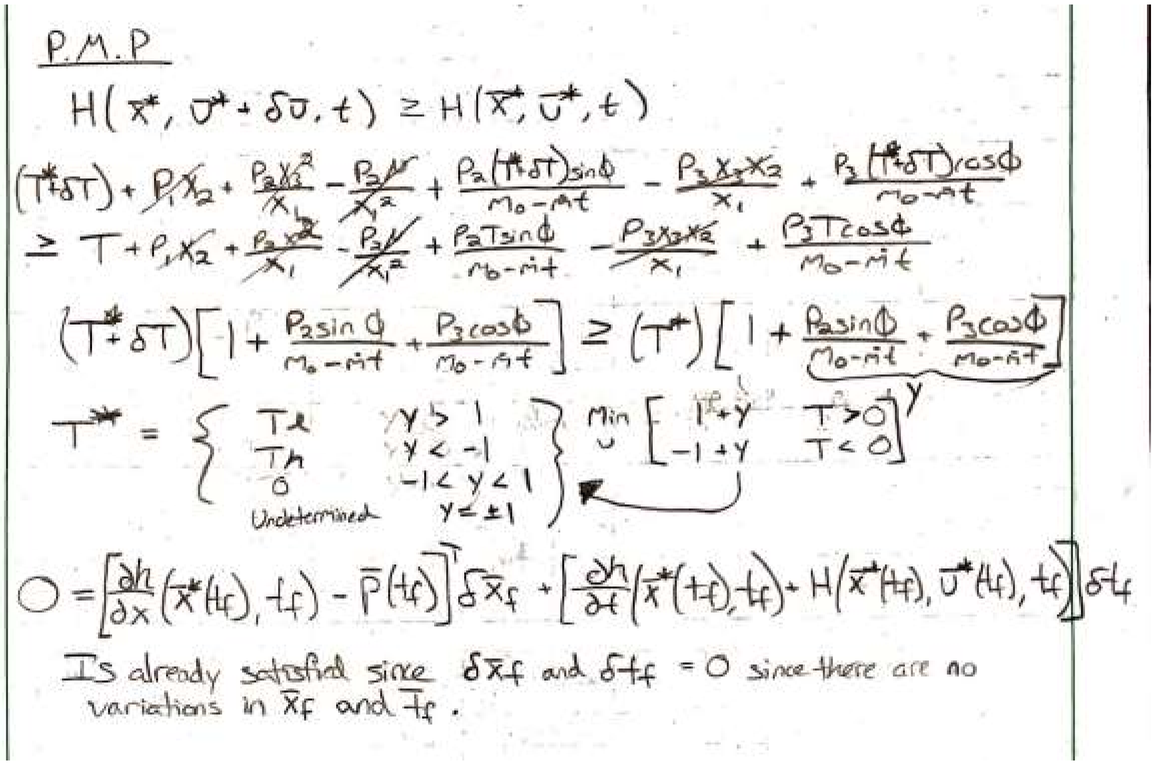
$$+P_2\cos\phi - P_3\sin\phi = 0$$

$$\tan\phi = -P_2/P_3$$

$$\phi = \tan^{-1}(P_2/P_3)$$

$$\frac{\partial H}{\partial T} = 1 + \frac{P_2\sin\phi}{m_0 - \dot{m}t} + \frac{P_3\cos\phi}{m_0 - \dot{m}t} = 0$$ (Not useful since T is linear so is not in this equation)

∴ Use P.M.P



P.M.P

$$H(\bar{x}^*, \bar{U}^* + \delta \bar{U}, t) \geq H(\bar{x}^*, \bar{U}^*, t)$$

$$(T^* + \delta T) + \cancel{P_1 x_2} + \cancel{\frac{P_2 x_3^2}{x_1}} - \cancel{\frac{P_2 \mu}{x_1^2}} + \frac{P_2 (T^* + \delta T)\sin\phi}{m_0 - \dot{m}t} - \cancel{\frac{P_3 x_3 x_2}{x_1}} + \frac{P_3 (T^* + \delta T)\cos\phi}{m_0 - \dot{m}t}$$

$$\geq T + \cancel{P_1 x_2} + \cancel{\frac{P_2 x_3^2}{x_1}} - \cancel{\frac{P_2 \mu}{x_1^2}} + \frac{P_2 T \sin\phi}{m_0 - \dot{m}t} - \cancel{\frac{P_3 x_3 x_2}{x_1}} + \frac{P_3 T \cos\phi}{m_0 - \dot{m}t}$$

$$(T^* + \delta T)\left[1 + \frac{P_2 \sin\phi}{m_0 - \dot{m}t} + \frac{P_3 \cos\phi}{m_0 - \dot{m}t}\right] \geq (T^*)\left[1 + \underbrace{\frac{P_2 \sin\phi}{m_0 - \dot{m}t} + \frac{P_3 \cos\phi}{m_0 - \dot{m}t}}_{Y}\right]$$

$$T^* = \begin{cases} T_l & Y > 1 \\ T_h & Y < -1 \\ 0 & -1 < Y < 1 \\ \text{Undetermined} & Y = \pm 1 \end{cases} \xleftarrow{} \min_{U} \begin{bmatrix} 1 + Y & T > 0 \\ -1 + Y & T < 0 \end{bmatrix}$$

$$0 = \left[\frac{\partial h}{\partial x}(\bar{x}^*(t_f), t_f) - \bar{P}(t_f)\right]^T \delta \bar{x}_f + \left[\frac{\partial h}{\partial t}(\bar{x}^*(t_f), t_f) + H(\bar{x}^*(t_f), \bar{U}^*(t_f), t_f)\right]\delta t_f$$

Is already satisfied since $\delta \bar{x}_f$ and $\delta t_f = 0$ since there are no variations in $\bar{x}_f$ and t_f.

Appendix A.2: MATLAB Code

Variation of Extremals with PMP for Low Thrust from LEO to GEO

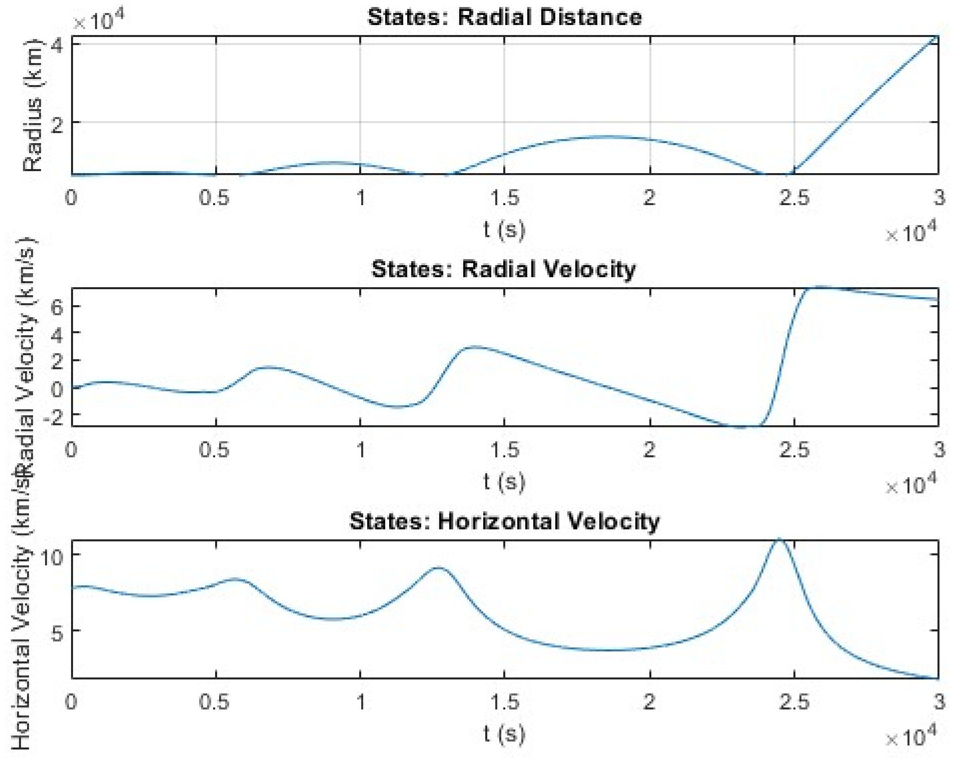
Devin Quach

```
clc
clear all
close all

mu = 398600.5;                          % In km^3 /s^2
% === Desired Final State === %
TOF = 30002; %9*60*60;                          % 9 Hours In Seconds
r_final = 35786+6378;                   % Final Radius: kMeters
u_final = 0 ;                           % Final Radial Velocity:km/s
v_final = sqrt(mu/r_final);             % Final Horizontal Velocity: km/s

% === Physical Constants === %
mdot = 0.1;                          % In kg/s Assumed to be Constant
mo = 3000; %11600;                 % In kg (Mass of Hubble Telescope)


% === Computational Variables ==== %
NumTSteps = TOF;                         % # of TimeSteps
Deltat = TOF/NumTSteps;                  % Time Increments for # of time steps till TOF
t = 0:Deltat:TOF;
tol = 0.001;                              % End Condition
NumXVars = 3;

% === Weighting/Constants === %
tau = 2;                               % Weighting on New Control
del = 1e-3;                              % Pertubation Size for Inital Costates

% === Preallocating === %
X = ones(NumTSteps+1,NumXVars);         % Preallocating State Vector
a = zeros(NumTSteps+1,NumXVars);        % Preallocating EOM Vector
P = ones(NumTSteps+1,NumXVars)*(10e3);   % Initial Guess for P & Preallocating
Pdot = zeros(NumTSteps+1,NumXVars);     % Preallocating Costate Dot Vector
P(1,:) = [5-4.6833462165128,-857.693225185878,-4804.96591508175];  %From Dr.T's max radius problem
% P(1,:) = [-1.56004780670084,2461.87889706418,-3223.51891396842];

% Assumption Givens for in a Vacumm (Merlin 1D)
% === Physical Givens === %
r = 160+6378 ;                          % Initial Radius: kMeters
u = 0 ;                                 % Initial Radial Velocity: km/s
v = sqrt(mu/r);                         % Initial Horizontal Velocity: km/s
X(1,:) = [r, u, v];                     % Initial State Vector

Thigh =1;% 981/mo;%981.5;                          % Higher Bound Constraint of Thrust Acceleration in
km/s^2
Tlow = 0;                    % Lower Bound Constraint of Thrust in kN

iter = 1;                               % Start of iterations
```

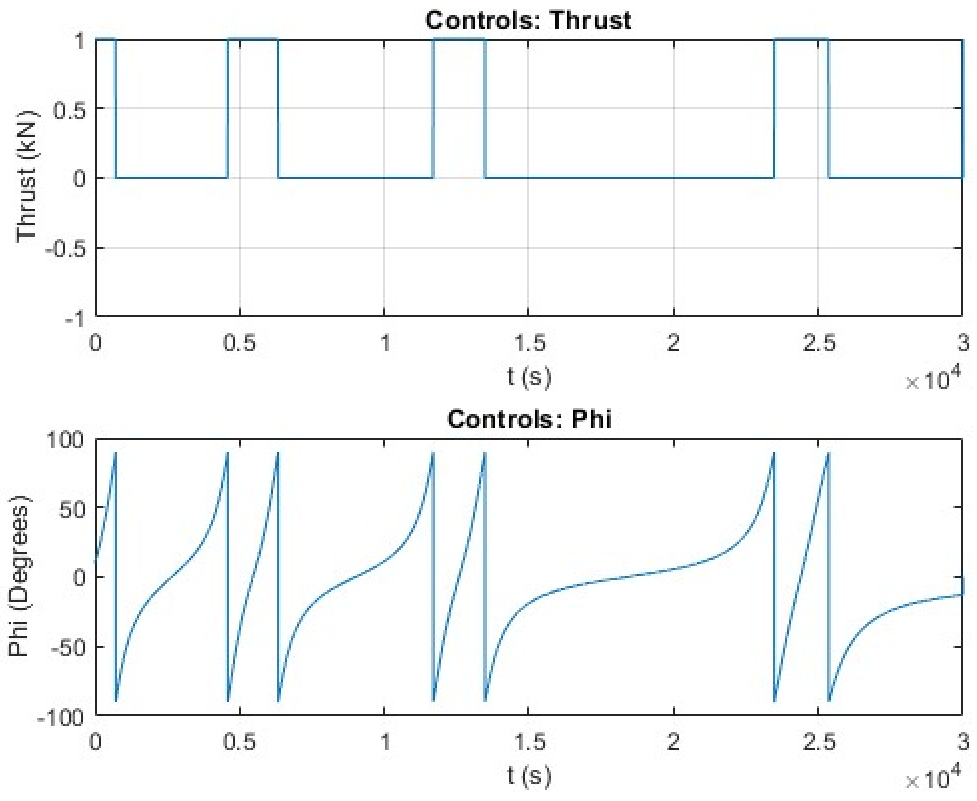
```
while norm( X(end,:) - [r_final, u_final, v_final] ) > tol
        % === Reduced Diff. Eqs & Propogate Costate Guess Forward to Find Final States=== %
        for k = 1:1:NumTSteps                    % Uses initial guess for P to perturb EOM vector and
Pdot, then use Euler's Iteration to solve for a new X and P


            u(k,2) = atan( P(k,2)/P(k,3) );
% Control for Phi

            if ((P(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((P(k,3)*cos(u(k,2)))/(mo-(mdot*t(k)))) > 1
% PMP Piecewise Control for T
               u(k,1) = Tlow;
            elseif  ((P(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((P(k,3)*cos(u(k,2)))/(mo-(mdot*t(k)))) <
-1
               u(k,1) = Thigh;
            else %((P(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((P(k,3)*cos(u(k,2)))/(mo-(mdot*t(k)))) < 1
&& ((P(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((P(k,3)*cos(u(k,2)))/(mo-(mdot*t(k)))) > -1
               u(k,1) = 0;
            end

            a(k,:) = [ X(k,2), ...
                (X(k,3)^2/X(k,1)) - (mu/(X(k,1))^2) + ((u(k,1)*sin(u(k,2)))/(mo-(mdot*t(k)))),...
                (-(X(k,3)*X(k,2))/(X(k,1))) + ((u(k,1)*cos(u(k,2)))/(mo-(mdot*t(k)))) ];
% EOM Vector

            X(k+1,:) = a(k,:)*Deltat + X(k,:);
% New State Vector using Forward Integration

            Pdot(k,:) = [ ((P(k,2)*(X(k,3))^2)/(X(k,1)^2)) - ((P(k,3)*X(k,3)*X(k,2))/(X(k,1)^2)) -
((2*P(k,2)*mu)/(X(k,1)^3)),...
                -P(k,1) + ((P(k,3)*X(k,3))/(X(k,1))),...
                -((2*P(k,2)*X(k,3))/X(k,1)) + ((P(k,3)*X(k,2))/(X(k,1))) ];
% CoState Dot Vector

            P(k+1,:) = Pdot(k,:)*Deltat + P(k,:);
% New CoState Vector using Forward Integration
        end

        % === Part C: Influence Fcns === % Using Finite Difference

        for m = 1:1:3                                               % Column Wise
            Ppert(1,:) = P(1,:);
            Xpert(1,:) = X(1,:);
            Ppert(1,m) = (P(1,m)+del);
                for k = 1:1:NumTSteps     % Uses initial guess for P to make perturb a vector, then use
Euler's Iteration to solve for a new X

                    u(k,2) = atan( Ppert(k,2)/Ppert(k,3) );
% Control for Phi

                    if ((Ppert(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((Ppert(k,3)*cos(u(k,2)))/(mo-
(mdot*t(k)))) > 1     % PMP Piecewise Control for T
                       u(k,1) = Tlow;
                    elseif  ((Ppert(k,2)*sin(u(k,2)))/(mo-(mdot*t(k)))) + ((Ppert(k,3)*cos(u(k,2)))/(mo-
(mdot*t(k)))) < -1
                        u(k,1) = Thigh;
                    else
                        u(k,1) = 0;
                    end

            apert(k,:) = [ Xpert(k,2), ...
                (Xpert(k,3)^2/Xpert(k,1)) - (mu/(Xpert(k,1))^2) + ((u(k,1)*sin(u(k,2)))/(mo-
(mdot*t(k)))),...
                (-(Xpert(k,3)*Xpert(k,2))/(Xpert(k,1))) + ((u(k,1)*cos(u(k,2)))/(mo-(mdot*t(k)))) ];
% EOM Vector

            Xpert(k+1,:) = apert(k,:)*Deltat + Xpert(k,:);
% New State Vector using Forward Integration

            Pdotpert(k,:) = [ ((Ppert(k,2)*(Xpert(k,3))^2)/(Xpert(k,1)^2)) -
((Ppert(k,3)*Xpert(k,3)*Xpert(k,2))/(Xpert(k,1)^2)) - ((2*Ppert(k,2)*mu)/(Xpert(k,1)^3)),...
                -Ppert(k,1) + ((Ppert(k,3)*Xpert(k,3))/(Xpert(k,1))),...
                -((2*Ppert(k,2)*Xpert(k,3))/Xpert(k,1)) + ((Ppert(k,3)*Xpert(k,2))/(Xpert(k,1))) ];
% CoState Dot Vector

            Ppert(k+1,:) = Pdotpert(k,:)*Deltat + Ppert(k,:);
% Perturbed CoState Vector using Forward Integration
                end
            Px(:,m) =   (Xpert(end,:)-X(end,:))' / del ;                      % Column of Px
        end

        P(1,:) = P(1,:) - tau*( (Px^-1)*( X(end,:) - [r_final, u_final, v_final] )')';        %
Propogating through to get New Costate Vector
iter = iter+1;
end
u(end+1,:) = [0 0];

z(:,1) = X(:,1);
z(:,2) = X(:,2);
z(:,3) = X(:,3);
z(:,4) = u(:,1);
z(:,5) = u(:,2)*(180/pi);
% === Plotting Results === %

figure(1)
subplot(3,1,1)
plot(t,X(:,1))
title('States: Radial Distance')
xlabel('t (s)')
ylabel('Radius (km)')
axis([0 TOF r-10 r_final+10])
grid on

subplot(3,1,2)
plot(t,X(:,2))
title('States: Radial Velocity')
xlabel('t (s)')
ylabel('Radial Velocity (km/s)')
```

```
axis([0 TOF min(X(:,2)) max(X(:,2)) ])

subplot(3,1,3)
plot(t,X(:,3))
title('States: Horizontal Velocity')
xlabel('t (s)')
ylabel('Horizontal Velocity (km/s)')
axis([0 TOF min(X(:,3)) max(X(:,3))] )

figure(2)
subplot(2,1,1)
plot(t(1:end),u(:,1))
title('Controls: Thrust')
xlabel('t (s)')
ylabel('Thrust (kN)')
grid on
axis([0 TOF -Thigh Thigh ])

subplot(2,1,2)
plot(t(1:end),u(:,2)*180/pi)
title('Controls: Phi')
xlabel('t (s)')
ylabel('Phi (Degrees)')
axis([0 TOF -100 100] )
```

```
Warning: Matrix is singular,
close to singular or badly
scaled. Results may be
inaccurate. RCOND = NaN.
```



States: Radial Distance
Radius (km)
×10^4
t (s)
States: Radial Velocity
Radial Velocity (km/s)
t (s)
States: Horizontal Velocity
Horizontal Velocity (km/s)
t (s)
0
0.5
1
1.5
2
2.5
3
×10^4



Controls: Thrust
Thrust (kN)
t (s)
Controls: Phi
Phi (Degrees)
t (s)
×10^4

Published with MATLAB® R2018a