EML6934 Midterm Exam

Elias Reyes

April 7, 2022

1 Differential Equations of Motion

The equations of motion were derived using both Newton's second law and Lagrange's equations. The schematic for the problem can be seen Figure 1. The spacecraft is modeled as point P of mass m. The spacecraft moves relative to an inertial reference frame l. The reference frame fixed in l is expressed as $\{e_x, e_y, e_z\}$. The position of the spacecraft is denoted as $r_{P/O}$, where O is modeled as the sun, fixed in l. The spacecraft is parameterized in the basis $\{u_r, u_\theta, u_z\}$, where the rotation is about $u_z = e_z$. The rotation creates an angle θ between e_x and u_r, which can be seen in Figure 2.

Two forces are said to act on the spacecraft. The first is the gravitational force which is given as

$$G = -m\mu \frac{r_{P/O}}{||r_{P/O}||^3}, \tag{1}$$

while the second is the thrust force given as

$$T = Tw, \tag{2}$$

where w is the unit vector that lies an angle β from the direction u_θ as seen in Figure 3.

1.1 Position, Velocity and Acceleration of the Spacecraft

As seen in Figure 1, the position of the spacecraft represented in reference frame A, is given as

$${}^A\vec{r}_{P/O} = ru_r. \tag{3}$$

The velocity can then be represented in the inertial frame l by using equation 4, where ${}^l\vec{\omega}^A$ is the angular velocity between reference frame l and A.

$$\frac{{}^l d\vec{r}_{P/O}}{dt} = \frac{{}^A d\vec{r}_{P/O}}{dt} + {}^l\vec{\omega}^A \times {}^A\vec{r}_{P/O} \tag{4}$$

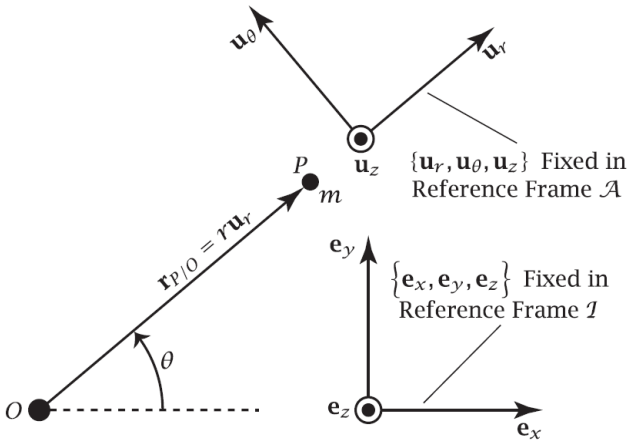

Figure 1: Schematic of particle moving in an inertially fixed plane

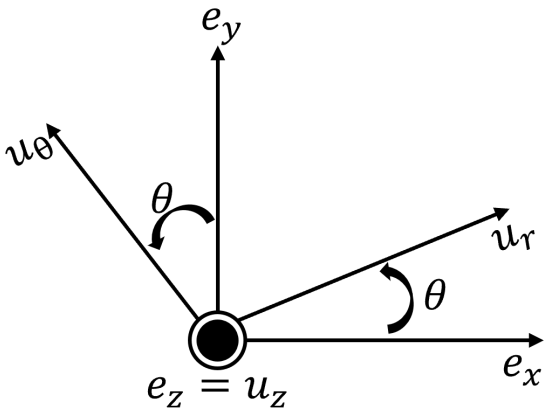

Figure 2: Reference frame rotation

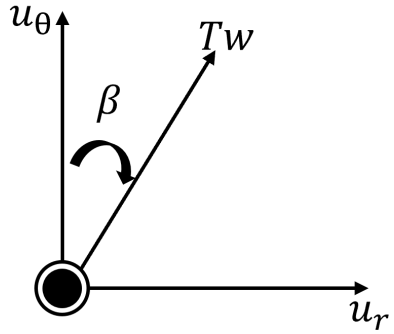

Figure 3: Thrust Force at Angle β

Using Equation 4, the velocity of the spacecraft in the inertial frame is then formulated as

$$\begin{aligned} {}^{l}\vec{v_p} &= \frac{{}^{l}d\vec{r}_{P/O}}{dt} \\ {}^{l}\vec{v_p} &= \dot{r}u_r + \dot{\theta}u_z \times ru_r \\ {}^{l}\vec{v_p} &= \dot{r}u_r + \dot{\theta}ru_\theta \end{aligned} \tag{5}$$

The acceleration of the particle can then be formulated as

$$\begin{aligned} {}^{l}\vec{a_p} &= \frac{{}^{l}d\vec{v_P}}{dt} \\ {}^{l}\vec{a_p} &= \frac{{}^{A}d\vec{v_P}}{dt} + {}^{l}\vec{\omega}^{A} \times {}^{A}\vec{v_P} \\ {}^{l}\vec{a_p} &= \ddot{r}u_r + (\ddot{\theta}r + \dot{\theta}\dot{r})u_\theta + \dot{\theta}\dot{r}u_\theta - \dot{\theta}^2 ru_r \\ {}^{l}\vec{a_p} &= (\ddot{r} - \dot{\theta}^2 r)u_r + (\ddot{\theta}r + 2\dot{\theta}\dot{r})u_\theta \end{aligned} \tag{6}$$

1.2 Newton's Second Law for A Particle

The equations of motion are first derived using Newton's Second Law for a Particle. Newton's Second Law for a particle is represented by

$$\sum F_P = m_P * a_P. \tag{7}$$

Figure 4 represents the free body diagram of the particle system. F_G, represented by equation 1, is the gravitational force and acts along the u_r direction. F_T, represented by equation 2, is the thrust force and acts in the direction w. It can be seen in Figure 3 that the thrust force can be re-written as

$$F_T = Tsin(\beta)u_r + Tcos(\beta)u_\theta, \tag{8}$$

while the gravitational force can be written as

$$F_G = -m\mu\frac{1}{r^2}u_r. \tag{9}$$

We can now substitute equations 6, 8, and 9 into equation 7 to formulate

$$-m\mu\frac{1}{r^2}u_r + Tsin(\beta)u_r + Tcos(\beta)u_\theta = m[(\ddot{r} - \dot{\theta}^2 r)u_r + (\ddot{\theta}r + 2\dot{\theta}\dot{r})u_\theta].$$

After equating terms, the two equations of motion using Newtons Seconds become:

$$(u_r) \qquad \ddot{r} = \dot{\theta}^2 r - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m} \tag{10}$$

$$(u_\theta) \qquad \ddot{\theta} = -\frac{2\dot{r}\dot{\theta}}{r} + \frac{Tcos(\beta)}{mr} \tag{11}$$

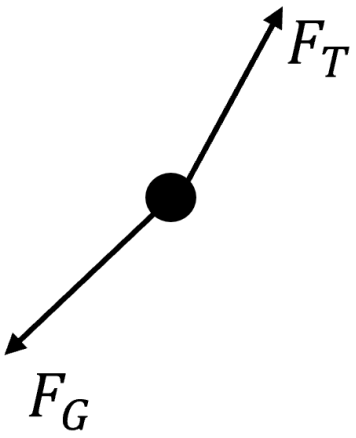

Figure 4: Free Body Diagram

1.3 Lagrange's Equations

The equations of motion are then derived using the Standard Form of Lagrange's Equations. The Standard Form of Lagrange's Equation is represented as

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = Q^{'}, \tag{12}$$

where L is the Lagrangian that can be represented as

$$L = T - V. \tag{13}$$

Here, T is the kinetic energy and V is the scalar potential. The kinetic energy for this system is calculated as

$$\begin{aligned} T &= \frac{1}{2} m v_P \cdot v_P, \\ T &= \frac{1}{2} m[(\dot{r}u_r + \ \dot{\theta} r u_\theta) \cdot (\dot{r}u_r + \ \dot{\theta} r u_\theta)], \\ T &= \frac{1}{2} m(\dot{r}^2 + \dot{\theta}^2 r^2), \end{aligned} \tag{14}$$

while the scalar potential is calculated as

$$V = -\frac{m\mu}{r}. \tag{15}$$

Substituting equations 14 and 15 into equation 13, the Lagrangian becomes

$$L = \frac{1}{2} m(\dot{r}^2 + \dot{\theta}^2 r^2) + \frac{m\mu}{r}. \tag{16}$$

The generalized coordinates for this system are $q_1 = r$ and $q_2 = \theta$. The derivatives in equation 12, for each generalized coordinate, can then be expressed as:

$$\frac{\partial L}{\partial q_1} = m\dot{\theta}^2 r - \frac{m\mu}{r}, \qquad \frac{\partial L}{\partial \dot{q}_1} = m\dot{r}, \qquad \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_1} = m\ddot{r},$$
$$\frac{\partial L}{\partial q_2} = 0, \qquad \frac{\partial L}{\partial \dot{q}_2} = m\dot{\theta}r^2, \qquad \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_2} = m\ddot{\theta}r^2 + 2mr\dot{\theta}\dot{r}.$$

The generalized forces that are not derivable from a scalar potential function, for the generalized coordinate r, are then formulated as:

$$\begin{aligned} Q_1^{'} &= Tw \cdot \frac{\partial r_{P/O}}{\partial r}, \\ Q_1^{'} &= [Tcos(\beta)u_\theta + Tsin(\beta)u_r] \cdot \frac{\partial r u_r}{\partial r}, \\ Q_1^{'} &= Tsin(\beta). \end{aligned} \tag{17}$$

The generalized forces that are not derivable from a scalar potential function, for the generalized coordinate θ, are then formulated as:

$$\begin{aligned} Q_2^{'} &= Tw \cdot \frac{\partial r_{P/O}}{\partial r}, \\ Q_2^{'} &= [Tcos(\beta)u_\theta + Tsin(\beta)u_r] \cdot \frac{\partial r u_r}{\partial \theta}, \end{aligned}$$

where $u_r = cos(\theta)e_x + sin(\theta)e_y$. Then,

$$\begin{aligned} Q_2^{'} &= [Tcos(\beta)u_\theta + Tsin(\beta)u_r] \cdot \frac{\partial(rcos(\theta)e_x + rsin(\theta)e_y)}{\partial \theta}, \\ Q_2^{'} &= [Tcos(\beta)u_\theta + Tsin(\beta)u_r] \cdot r(-sin(\theta)e_x + cos(\theta)e_y), \end{aligned}$$

where $u_\theta = -sin(\theta)e_x + cos(\theta)e_y$. Then,

$$\begin{aligned} Q_2^{'} &= [Tcos(\beta)u_\theta + Tsin(\beta)u_r] \cdot ru_\theta, \\ Q_2^{'} &= Tcos(\beta)r. \end{aligned} \tag{18}$$

Then, after substituting equations 17, 18, and the derivatives, into equation 12, the two equations of motion become:

$$\begin{aligned} m\ddot{r} - m\dot{\theta}^2 r + \frac{m\mu}{r} &= Tsin(\beta), \\ m\ddot{r} &= m\dot{\theta}^2 r - \frac{m\mu}{r} + Tsin(\beta), \\ \ddot{r} &= \dot{\theta}^2 r - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m}, \end{aligned} \tag{19}$$

and,

$$\begin{aligned} m\ddot{\theta}r^2 + 2mr\dot{\theta}\dot{r} &= Tcos(\beta)r, \\ m\ddot{\theta}r^2 &= -2mr\dot{\theta}\dot{r} + Tcos(\beta)r, \\ \ddot{\theta} &= -\frac{2\dot{r}\dot{\theta}}{r} + \frac{Tcos(\beta)}{mr}. \end{aligned} \tag{20}$$

Equations 10 and 11 represent the equations of motion derived using Newtons Second Law, while equations 19 and 20 represent the equations of motion derived using Lagrange's Equations.

1.4 Conversion to First-Order Equations

To re-write the two second-order equations into first four first-order equations, the following substitutions can be made:

$$\dot{r} = v_r, \tag{21}$$

$$\begin{aligned} r\dot{\theta} &= v_\theta, \\ \dot{\theta} &= \frac{v_\theta}{r}. \end{aligned} \tag{22}$$

Then, taking the derivatives of r and θ of equations 21 and 22, respectively:

$$\ddot{r} = \dot{v}_r, \tag{23}$$

$$\ddot{\theta} = \frac{\dot{v}_\theta r - v_\theta \dot{r}}{r^2} \tag{24}$$

After substituting equations 21, 22, 23, and 24 into equations 19 and 20, two first-order equations are derived as:

$$\begin{aligned} \dot{v}_r &= \frac{v_\theta^2 r}{r^2} - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m}, \\ \dot{v}_r &= \frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m}, \end{aligned} \tag{25}$$

and,

$$\begin{aligned} \frac{\dot{v}_\theta r - v_\theta v_r}{r^2} &= -\frac{2v_\theta v_r}{r^2} + \frac{Tcos(\beta)}{mr}, \\ \dot{v}_\theta r - v_\theta v_r &= -\frac{2v_\theta v_r r^2}{r^2} + \frac{Tcos(\beta)r^2}{mr}, \\ \dot{v}_\theta &= -\frac{2v_\theta v_r}{r} + \frac{Tcos(\beta)}{m} + \frac{v_\theta v_r}{r}, \\ \dot{v}_\theta &= -\frac{v_\theta v_r}{r} + \frac{Tcos(\beta)}{m}. \end{aligned} \tag{26}$$

Equations 21, 22, 25, and 26 are the four first-order equations derived from the two second-order equations 19 and 20. Lastly, a fifth first-order equations is given as:

$$\dot{m} = -\frac{T}{v_e} \tag{27}$$

2 Formulation of Optimal Control Problem

The objective of the optimal control problem is to minimize the time to transfer from an initial circular orbit to a final circular orbit. Because the thrust is constant, minimizing the time is equivalent to maximizing the final mass or minimizing the fuel burned. For simplicity, optimality conditions will be derived with the objective function being to maximize fuel at the terminal time. Below is an overview of the problem:

$$\begin{aligned} Objective: &\quad max\ m(t_f) \\ State: &\quad r(t), \theta(t), v_r(t), v_\theta(t), m(t) \\ Control: &\quad \beta(t) \end{aligned}$$

2.1 Boundary and First-Order Optimality Conditions

Since it is assumed that the spacecraft starts and ends in a circular orbit, the following boundary conditions at the initial and final time can be stated:

$$\begin{bmatrix} r(t_0) \\ \theta(t_0) \\ v_r(t_0) \\ v_\theta(t_0) \\ m(t_0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \sqrt{\frac{\mu}{r_0}} \\ 1 \end{bmatrix},$$

and,

$$\begin{bmatrix} r(t_f) \\ \theta(t_f) \\ v_r(t_f) \\ v_\theta(t_f) \\ m(t_f) \end{bmatrix} = \begin{bmatrix} 1 \\ free \\ 0 \\ \sqrt{\frac{\mu}{r_f}} \\ free \end{bmatrix}.$$

Since there are 5 state equations, there also exists 5 co-state equations. The co-state equations are all unknown at t_0 and can be expressed as followed:

$$\begin{bmatrix} \lambda_r(t_0) \\ \lambda_\theta(t_0) \\ \lambda_{v_r}(t_0) \\ \lambda_{v_\theta}(t_0) \\ \lambda_m(t_0) \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}.$$

The final time, t_f is also unknown, therefore, making that a total of 6 unknowns. Since there are 6 unknowns, there must also be 6 known conditions att_f. The remaining conditions are formulated by transversality equations. Before the transversality conditions are determined, the Hamiltonian must be introduced. The Hamiltonian can be calculated using the following equation:

$$H = L + \lambda^T f, \tag{28}$$

where $L = 0$, since $J = M$. Then, the Hamiltonian becomes

$$H = \lambda_r v_r + \lambda_\theta \frac{v_\theta}{r} + \lambda_{v_r}(\frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{Tsin(\beta)}{m}) + \lambda_{v_\theta}(-\frac{v_\theta v_r}{r} + \frac{Tcos(\beta)}{m}) + \lambda_m(-\frac{T}{v_e}).$$

The co-state equations can then be calculated as followed:

$$\dot{\lambda}_r = -\frac{\partial H}{\partial r} = \lambda_\theta \frac{v_\theta}{r^2} + \lambda_{v_r}(\frac{v_\theta^2}{r^2} - \frac{2\mu}{r^3}) - \lambda_{v_\theta}\frac{v_\theta v_r}{r^2}, \tag{29}$$

$$\dot{\lambda}_\theta = -\frac{\partial H}{\partial \theta} = 0, \tag{30}$$

$$\dot{\lambda}_{v_r} = -\frac{\partial H}{\partial v_r} = -\lambda_r + \lambda_{v_\theta}\frac{v_\theta}{r}, \tag{31}$$

$$\dot{\lambda}_{v_\theta} = -\frac{\partial H}{\partial v_\theta} = -\frac{\lambda_\theta}{r} - \frac{2\lambda_{v_r} v_\theta}{r} + \frac{\lambda_{v_\theta} v_r}{r}, \tag{32}$$

$$\dot{\lambda}_m = -\frac{\partial H}{\partial m} = \lambda_{v_r}\frac{Tsin(\beta)}{m^2} + \lambda_{v_\theta}\frac{Tcos(\beta)}{m^2}. \tag{33}$$

The optimal control can then be formulated as:

$$\begin{aligned} \frac{\partial H}{\partial \beta} &= \frac{\lambda_{v_r} Tcos(\beta)}{m} - \frac{\lambda_{v_\theta} Tsin(\beta)}{m} = 0, \\ \frac{\lambda_{v_r} Tcos(\beta)}{m} &= \frac{\lambda_{v_\theta} Tsin(\beta)}{m}, \\ \beta &= atan2(\frac{\lambda_{v_r}}{\lambda_{v_\theta}}). \end{aligned} \tag{34}$$

Now, since $\theta(t_f)$ and $m(t_f)$ are free, there exists transversality conditions on the co-state at t_f. Since $\delta t_0 = 0$, there are no transversailty conditions on the co-state at t_0. The conditions are as follows:

$$\lambda_\theta(tf) - \frac{\partial M}{\partial \theta(t_f)} + v^T \frac{\partial b}{\partial \theta(t_f)} = 0.$$

Since $M = m$, and due to there not being a boundary condition on $\theta(t_f)$, the transversality condition on the co-sate, λ_θ, becomes:

$$\lambda_\theta(tf) = 0. \tag{35}$$

Now,

$$\lambda_m(tf) - \frac{\partial M}{\partial m(t_f)} + v^T \frac{\partial b}{\partial m(t_f)} = 0.$$

Since $M = m$, and due to there not being a boundary condition on $\theta(t_f)$, the transversality condition on the co-sate, λ_m, becomes:

$$\lambda_m(tf) = 1. \tag{36}$$

Lastly, since t_f is free, $\delta t_f \neq 0$ and there is a transversality condition on the Hamiltonian at t_f as follows:

$$H|_{t_f} + \frac{\partial M}{\partial t_f} - v^T \frac{\partial b}{\partial t_f} = 0.$$

Since $M = m$, and due to there not being a boundary condition on t_f, the transversality condition on the Hamiltonian becomes:

$$H|_{t_f} = 0. \tag{37}$$

There are now a total of 6 conditions and 6 unknowns as listed below:

$$\left|\begin{array}{c} \underline{Conditions} \\ r(t_f) = 1.5 \\ v_r(t_f) = 0 \\ v_\theta(t_f) = \sqrt{\frac{\mu}{r_f}} \\ \lambda_\theta(t_f) = 0 \\ \lambda_m(t_f) = 1 \\ H|_{t_f} = 0 \end{array}\right|, \qquad \left|\begin{array}{c} \underline{Unknowns} \\ \lambda_r(t_0) \\ \lambda_\theta(t_0) \\ \lambda_{v_r}(t_0) \\ \lambda_{v_\theta}(t_0) \\ \lambda_m(t_0) \\ t_f \end{array}\right|.$$

2.2 Objective Functional

The objective functional is stated as

$$J = m(t_f) \tag{38}$$

2.3 Formal Statement of Optimal Control Problem

Determine the trajectory ($r(t)$, $\theta(t)$, $v_r(t)$, $v_\theta(t)$, $m(t)$) and the control $\beta(t)$ that maximizes the objective functional

$$J = m(t_f),$$

subject to the differential equation constraints:

$$
\begin{aligned}
\dot{r} &= v_r, \\
\dot{\theta} &= \frac{v_\theta}{r}, \\
\dot{v}_r &= \frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{T sin(\beta)}{m}, \\
\dot{v}_\theta &= -\frac{v_\theta v_r}{r} + \frac{T cos(\beta)}{m}, \\
\dot{m} &= -\frac{T}{v_e}, \\
\dot{\lambda}_r &= \lambda_\theta \frac{v_\theta}{r^2} + \lambda_{v_r}(\frac{v_\theta^2}{r^2} - \frac{2\mu}{r^3}) - \lambda_{v_\theta}\frac{v_\theta v_r}{r^2}, \\
\dot{\lambda}_\theta &= 0, \\
\dot{\lambda}_{v_r} &= -\lambda_r + \lambda_{v_\theta}\frac{v_\theta}{r}, \\
\dot{\lambda}_{v_\theta} &= -\frac{\lambda_\theta}{r} - \frac{2\lambda_{v_r} v_\theta}{r} + \frac{\lambda_{v_\theta} v_r}{r}, \\
\dot{\lambda}_m &= \lambda_{v_r}\frac{T sin(\beta)}{m^2} + \lambda_{v_\theta}\frac{T cos(\beta)}{m^2},
\end{aligned}
$$

and the boundary conditions:

$$
\begin{aligned}
r(t_0) = r_0 \quad &= 1, \\
\theta(t_0) = \theta_0 \quad &= 0, \\
v_r(t_0) = v_{r_0} \quad &= 0, \\
v_\theta(t_0) = v_{\theta_0} \quad &= \sqrt{\frac{\mu}{r_0}}, \\
m(t_0) = m_0 \quad &= 1, \\
r(t_f) = r_f \quad &= 1, \\
\theta(t_f) &= free, \\
v_r(t_f) = v_{r_f} \quad &= 0, \\
v_\theta(t_f) = v_{\theta_f} \quad &= \sqrt{\frac{\mu}{r_f}}, \\
m(t_f) &= free,
\end{aligned}
$$

and the transversality conditions:

$$
\begin{aligned}
\lambda_\theta(tf) &= 0, \\
\lambda_m(tf) &= 1, \\
H|_{t_f} &= 0,
\end{aligned}
$$

and the parameters:

$$\mu = 1,$$
$$T = 0.1405,$$
$$v_e = 1.8758344.$$

3 Numerical Solution of Optimal Control Problem

For the Numerical Solutions below, there are a couple things that should be noted. For the indirect methods, the objective function used was to maximize the fuel remaining at t_f ($J = M$). For the direct methods, the objective function was to minimize the final time, t_f. For this problem, since the thrust force is constant, maximizing the fuel remaining is essentially the same thing as minimizing the final time and minimizing the fuel consumption. Because of this, these three objectives are used interchangeably throughout the results and analysis. Another thing to emphasize is that the direct methods use a $\tau \in [-1\ 1]$ scale while the indirect methods use a time scale. This is due to the Matlab solvers having issues with the direct methods, specifically Direct Multiple Shooting.

3.1 Indirect Shooting

The first numerical method used to solve the optimal control problem was indirect shooting. Figure 5 shows the states for the trajectory that minimized the fuel consumption, or in other words, maximized the fuel remaining at t_f. Figure 6 shows the optimal control to achieve the optimal trajectory. Using Indirect Shooting, the final time was found to be 3.24694 while the final mass was optimized at 0.75550. For the simulation, the solution converged after 27 iterations and an elapsed time of 0.156104 seconds.

3.1.1 Analysis of Indirect Shooting

For this problem, the optimized terminal conditions were easily solvable using Indirect Shooting. The computation time was fast and the results were accurate. The only drawback to this method is that calculus of variations must be used, which is not always feasible. Also, for some problems, single shooting might push the limits of CPU due to large time steps. Luckily for this problem, that issue is not present.

Iterations (s)	Sim Time	Terminal Time	Terminal Mass
27	0.156104	3.24694	0.75550

Table 1: Performance for Indirect Shooting

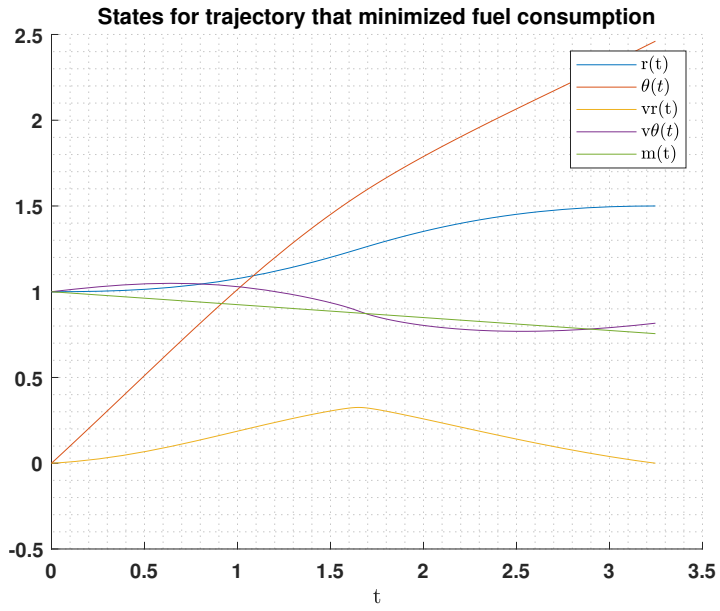

Figure 5: States for trajectory that minimized fuel consumption

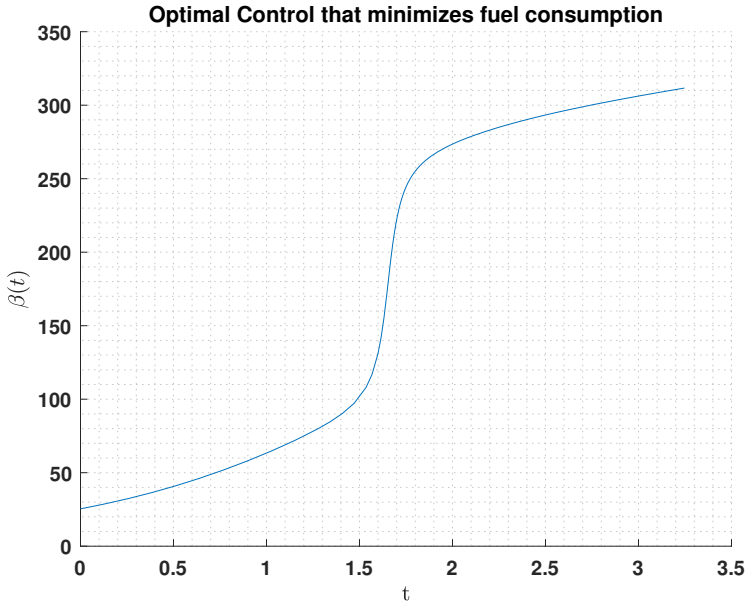

Figure 6: Optimal Control that minimized fuel consumption

3.2 Indirect Multiple Shooting

The second method used was Indirect Multiple Shooting, This method is much like Indirect Shooting, but unlike integrating over a single time interval, multiple shooting allows you to break the single interval up into smaller sub-intervals. This is generally done to reduce the magnitude of numbers worked with during optimization. For Indirect Multiple Shooting, the intervals tested were: $K = (2, 4, 8, 16)$.

Figure 7 shows the states for the trajectory that minimized the fuel consumption for intervals $K = 2$. Figure 8 shows the optimal control to achieve the optimal trajectory for intervals $K = 2$. The black asterisks in figures represent the start and end points of an interval. Much like Indirect Shooting, the optimized final time was found to be 3.24694 while the terminal mass was 0.7555. The solver converged after 16 iterations and an elapsed time of 0.5110 seconds.

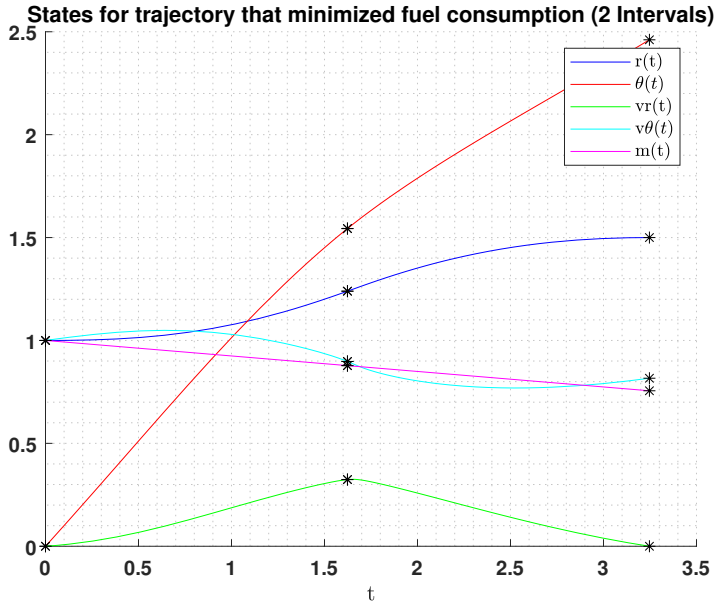

Figure 7: States for trajectory that minimized fuel consumption (2 intervals)

Figure 9 shows the states for the trajectory that minimized the fuel consumption for intervals $K = 4$, while Figure 10 shows the optimal control to achieve this trajectory. The optimized terminal conditions are the same as those found with interval $K = 4$. The terminal time was $t_f = 3.24694$ and the terminal mass was $m = 0.7555$. For 4 intervals, the solver needed 19 iterations and an elapsed time of 0.9880 seconds.

Figure 11 shows the states for the trajectory that minimized the fuel consumption for intervals $K = 8$, while Figure 12 shows the optimal control to achieve

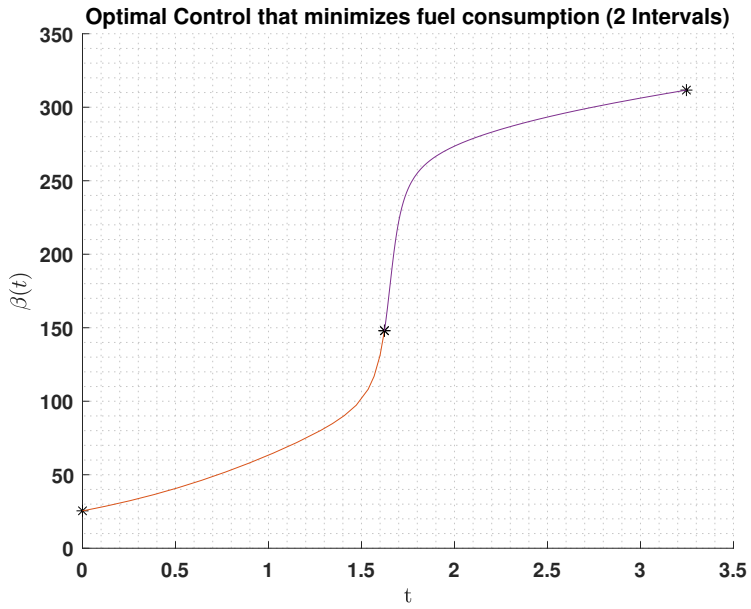

Figure 8: Optimal Control that minimized fuel consumption (2 intervals)

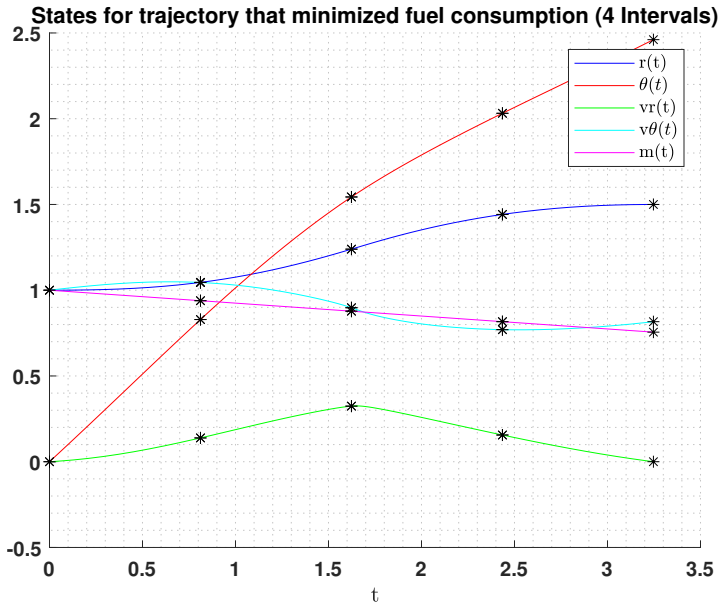

Figure 9: States for trajectory that minimized fuel consumption (4 intervals)

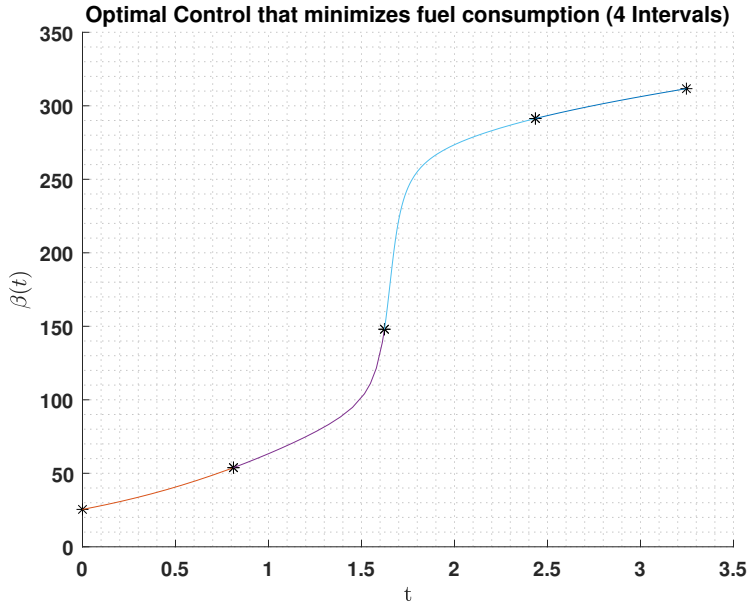

Figure 10: Optimal Control that minimized fuel consumption (4 intervals)

Intervals-K	Iterations (s)	Sim Time	Terminal Time	Terminal Mass
2	16	0.5110	3.24694	0.7555
4	19	0.9880	3.24694	0.7555
8	16	2.6119	3.24694	0.7555
16	19	10.9391	3.24694	0.7555

Table 2: Performance for Indirect Multiple Shooting

this trajectory. The optimized terminal time and mass are 3.24694 and 0.7555, respectively. For 8 intervals, the solution converged after 16 iterations and an elapsed time of 2.6119 seconds.

The final Indirect Multiple Shooting case is with intervals K = 16. Figure 13 shows the states for the trajectory that minimized the fuel consumption for intervals K = 16, while Figure 14 shows the optimal control to achieve this optimal trajectory. Just like all the other Indirect results, the terminal time and mass were optimized to be 3.246947 and 0.7555, respectively. However, for 16 interval the elapsed time increased to 10.9391 seconds. Lastly, the solution converged in 19 iterations.

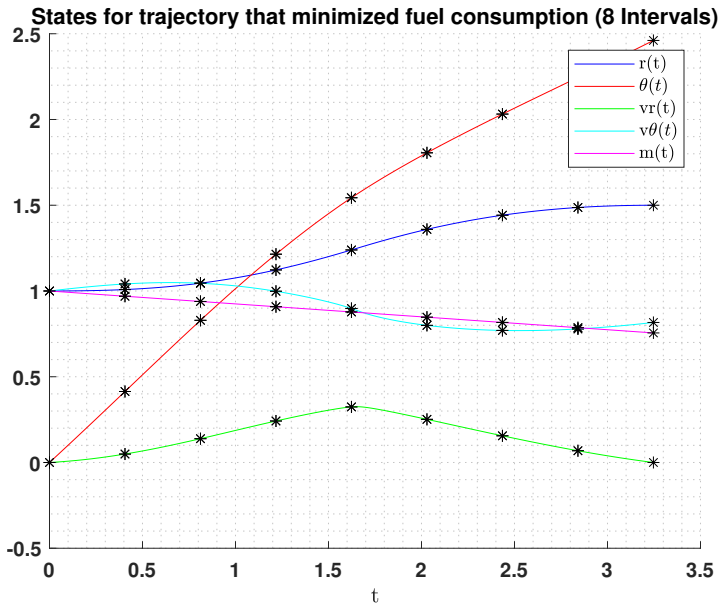

Figure 11: States for trajectory that minimized fuel consumption (8 intervals)

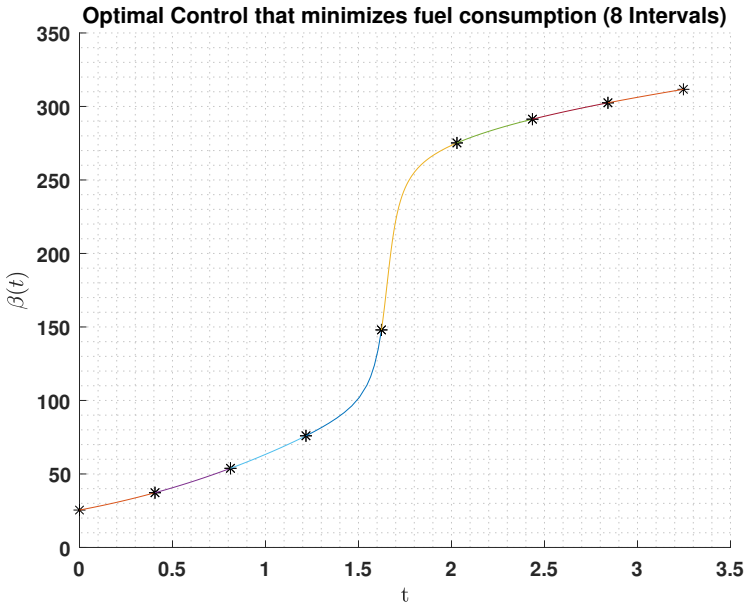

Figure 12: Optimal Control that minimized fuel consumption (8 intervals)

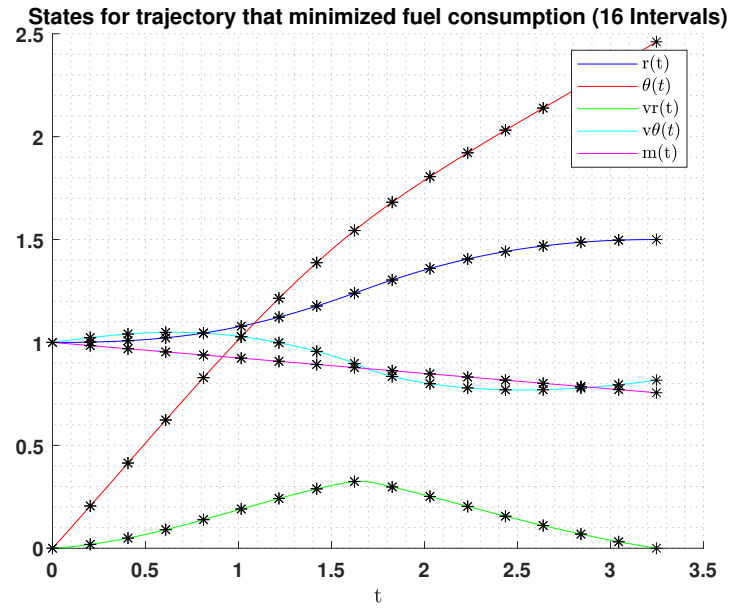

Figure 13: States for trajectory that minimized fuel consumption (16 intervals)

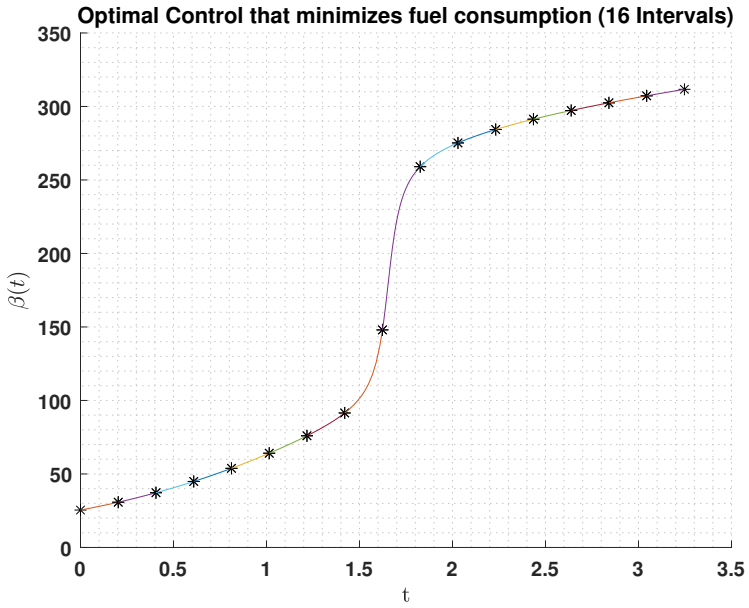

Figure 14: Optimal Control that minimized fuel consumption (16 intervals)

3.2.1 Analysis of Indirect Multiple Shooting

Interestingly, all of the Indirect Multiple Shooting cases achieved the same optimized terminal conditions as the Indirect Shooting Method. This is most likely due to the fact that the errors at the interval points are minuscule and that the optimal control can be directly computed from the Hamiltonian. For each case, regardless of the number of intervals, the optimal control seems to be continuous. The main different between all the cases is that the computation time increases as you include more intervals. For this particular problem, Indirect Multiple Shooting does not show benefit over Indirect Shooting due to the terminal conditions being the same and higher computation costs.

3.3 Direct Shooting

The third numerical method used is Direct Shooting. This method is much different than the indirect methods in the sense that calculus of variations is not employed. Instead, when solving the differential equations, the control is parameterized by linear independent basis functions. For this study, the basis functions are polynomials with degree N, ranging from 2 to 6. To get an optimized solution, the coefficients of the polynomials must be included in the optimization process. Also, as stated before, Direct Shooting was done on a τ scale, rather than on a time scale. This was done becuase the matlab solvers had issues optimizing Direct Multiple Shooting.

The first Direct Shooting case is with the control being parameterized by a polynomial of degree $N = 2$. Figure 15 shows the states for the trajectory that minimized the fuel consumption, while Figure 16 shows the optimal control to achieve this trajectory. Right away it is obvious that the results are much different from the previous Indirect Methods. This is because the control is approximated as a polynomial and it's obvious that from the Indirect Methods that the optimal control cant be well approximated with a single polynomial. For this case, single polynomial of degree $N = 2$, the optimized terminal time and mass were found to be 3.8662 and 0.7089, respectively. The solution converged after 13 iterations and an elapsed time of 0.1380 seconds.

The second Direct Shooting case is with the control being parameterized by a polynomial of degree $N = 3$. Figure 17 shows the states for the trajectory that minimized the fuel consumption, while Figure 18 shows the optimal control to achieve this trajectory. Again, it is obvious that the single polynomial does not represent the optimal control showed with the indirect methods. For this case, the optimized terminal time and mass were found to be 3.5109 and 0.7356, respectively. The solution converged after 23 iterations and an elapsed time of 0.2474 seconds.

The next Direct Shooting case is with the control being parameterized by a polynomial of degree $N = 4$. Figure 19 shows the states for the trajectory that minimized the fuel consumption, while Figure 20 shows the optimal control to achieve this trajectory. For this case, the optimized terminal time and mass

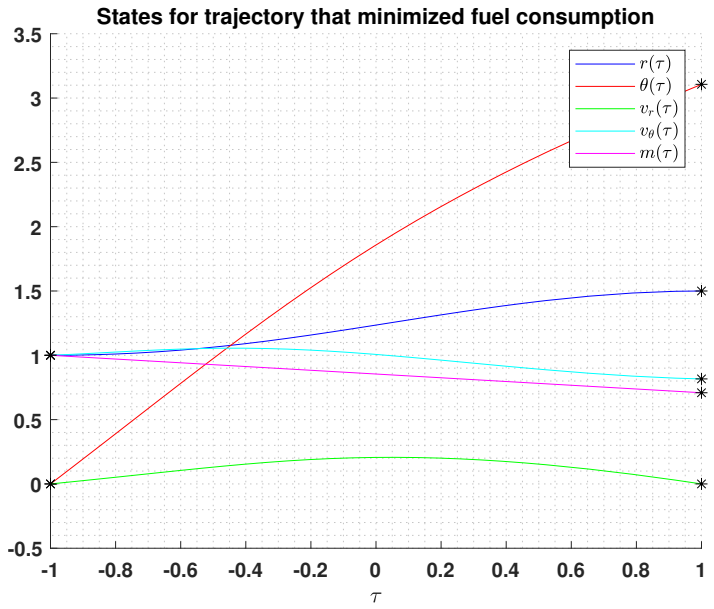

Figure 15: States for trajectory that minimized fuel consumption ($N = 2$)

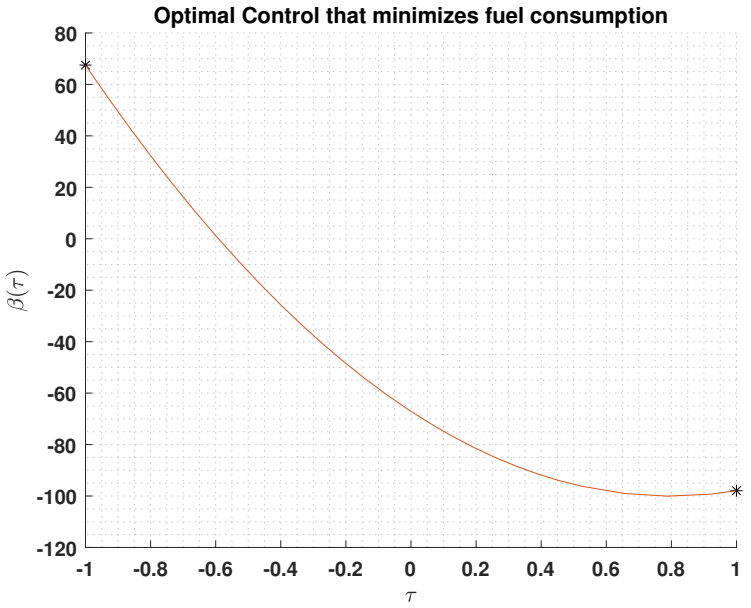

Figure 16: Optimal Control that minimized fuel consumption ($N = 2$)

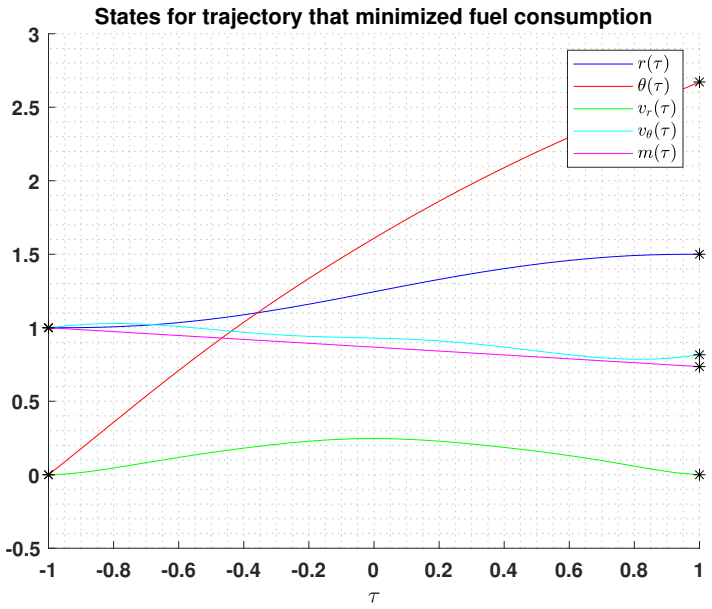

Figure 17: States for trajectory that minimized fuel consumption ($N = 3$)

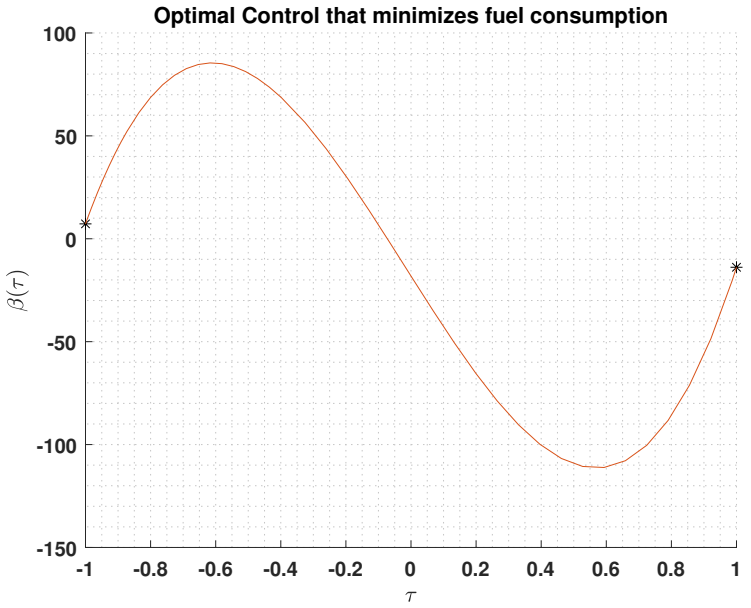

Figure 18: Optimal Control that minimized fuel consumption ($N = 3$)

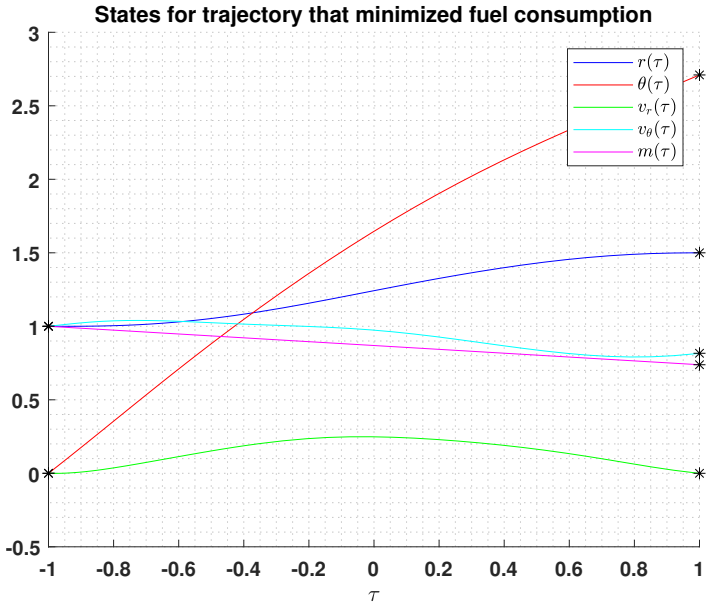

Figure 19: States for trajectory that minimized fuel consumption ($N = 4$)

were found to be 3.4723 and 0.73855, respectively. The solution converged after 26 iterations and an elapsed time of 0.4678 seconds.

The fourth Direct Shooting case is with the control being parameterized by a polynomial of degree $N = 5$. Figure 21 shows the states for the trajectory that minimized the fuel consumption, while Figure 22 shows the optimal control to achieve this trajectory. For this case, the optimized terminal time and mass were found to be 3.364 and 0.7466, respectively. The solution converged after 41 iterations and an elapsed time of 0.4678 seconds.

The final Direct Shooting case is with the control being parameterized by a polynomial of degree $N = 6$. Figure 23 shows the states for the trajectory that minimized the fuel consumption, while Figure 24 shows the optimal control to achieve this trajectory. For this case, the optimized terminal time and mass were found to be 3.361 and 0.74696, respectively. The solution converged after 51 iterations and an elapsed time of 0.6769 seconds.

3.3.1 Analysis of Direct Shooting

The Direct Shooting numerical method produced much different results than that of the two indirect methods. It is evident that the control being parameterized as a single polynomial basis function does not produce as optimized results as employing the calculus of variations. However, there are some highlights to the Direct Shooting method which are that it is much simpler to employ. Also,

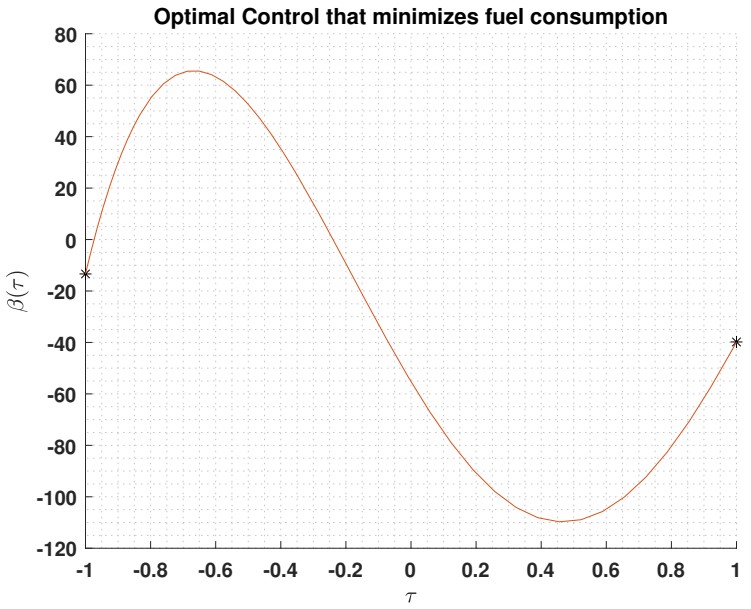

Figure 20: Optimal Control that minimized fuel consumption ($N = 4$)

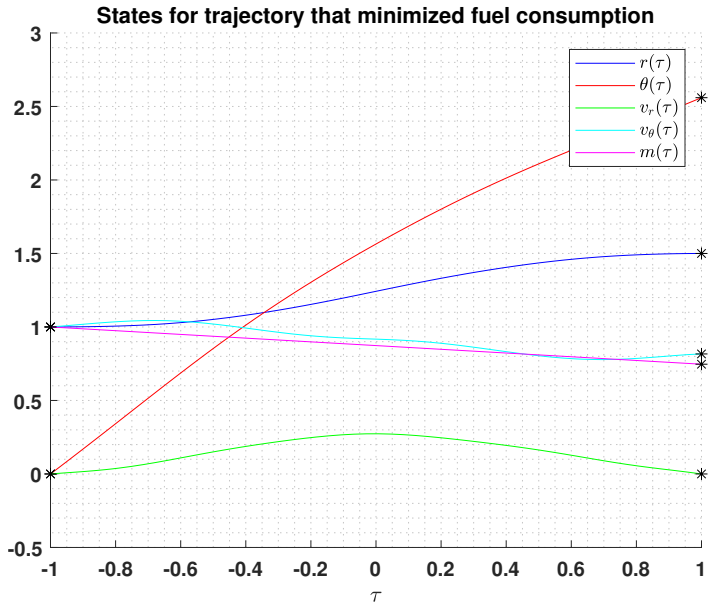

Figure 21: States for trajectory that minimized fuel consumption ($N = 5$)

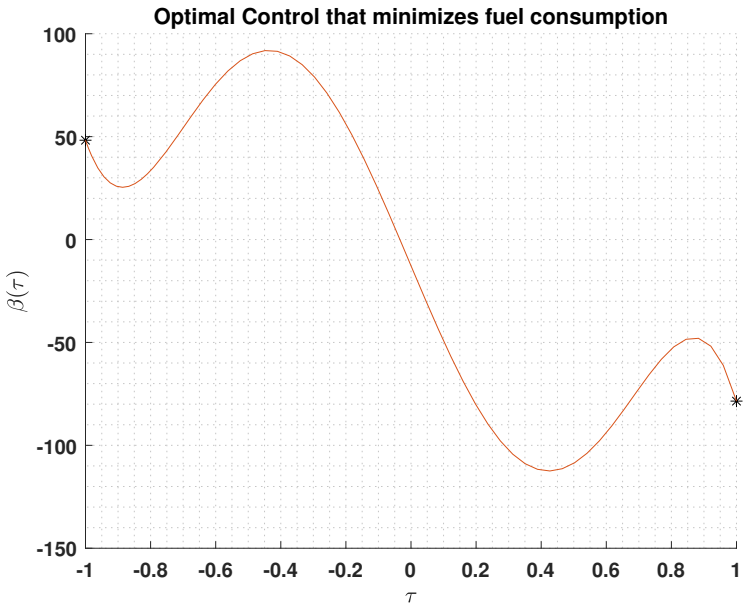

Figure 22: Optimal Control that minimized fuel consumption ($N = 5$)

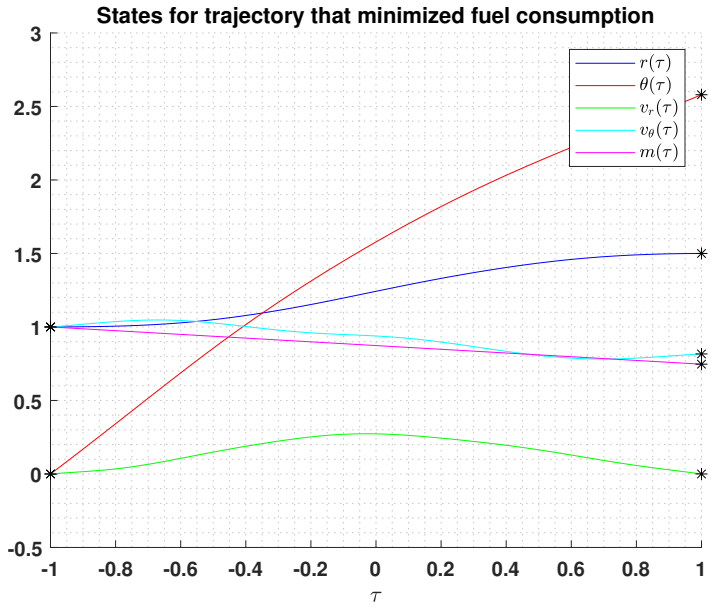

Figure 23: States for trajectory that minimized fuel consumption ($N = 6$)

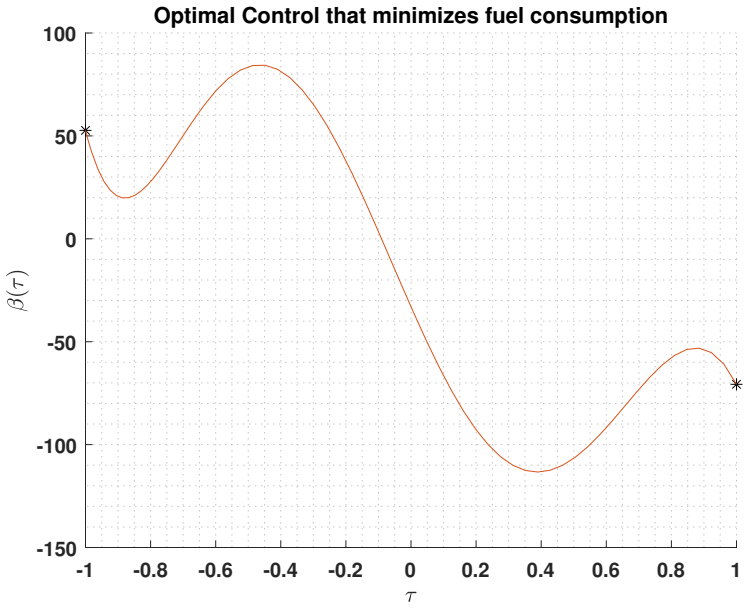

Figure 24: Optimal Control that minimized fuel consumption ($N = 6$)

Degree-N	Iterations (s)	Sim Time	Terminal Time	Terminal Mass
2	13	0.1380	3.8662	0.7089
3	23	0.2474	3.5109	0.7356
4	26	0.3590	3.4723	0.7385
5	41	0.4678	3.3641	0.7467
6	51	0.6769	3.3611	0.7469

Table 3: Performance for Direct Shooting

for single Direct Shooting case, the computation time is very efficient. Direct Shooting with polynomial degree $N = 2$, had an execution time of only 0.1380, which is lower than that of Indirect Shooting. The case with polynomial degree $N = 6$ was the most computationally heavy and only took 0.6769 to converge on a solution. While Indirect Shooting only took 0.1561 seconds, Indirect Multiple Shooting ranged from 0.511 to 10.94 seconds. Out of these methods the preferred method really comes down to the requirements. The largest terminal mass for the Direct Shooting method was 0.747 when the control is parameterized with a polynomial of degree $N = 6$. Depending on the scale, this could be a massive difference from the optimized terminal mass using the indirect methods. This results in an approximately 1% difference in mass.

3.4 Direct Multiple Shooting

The final numerical method used to solve the optimal control problem was Direct Multiple Shooting. Direct Multiple Shooting similar to Direct Shooting but utilized K intervals like Indirect Multiple Shooting. Direct Multiple Shooting allows for different sets of polynomial coefficients for each interval. For this study, Direct Multiple Shooting was performed for intervals $K = (2, 4, 8, 16)$ and with the control parameterized by polynomials of degree $N = (2, 3, 4, 5, 6)$.

The first set of Direct Multiple Shooting cases are performed with the degree polynomial $N = 2$ and intervals $K = (2, 4, 8, 16)$. Figure 25 shows the states for the trajectory that minimized the fuel consumption for $K = 2$ and $N = 2$, while Figure 26 shows the optimal control to achieve this trajectory. It is instantly noticeable that the optimal control is much closer to that of the indirect methods than any of the Direct Shooting cases. This is because having two intervals allows for the control to be parameterized with two different polynomials rather than one. For this case, the terminal time and mass were found to be 3.2489 and 0.755, respectively. The solution converged after 33 iterations and an elapsed time of 0.7972 seconds.

Figure 27 shows the states for the trajectory that minimized the fuel consumption for $K = 4$ and $N = 2$, while Figure 28 shows the optimal control to achieve this trajectory. For this case, the optimized terminal time and mass converged at 3.248 and 0.755, respectively. The solution converged after 64 iterations and an elapsed time of 3.0869 seconds. Already, it is noticeable that the optimized terminal conditions are very similar to those using the indirect methods. While the optimized conditions are similar, the computational cost are a little higher.

Figure 29 shows the states for the trajectory that minimized the fuel consumption for $K = 8$ and $N = 2$, while Figure 30 shows the optimal control to achieve this trajectory. Similar to two intervals, four intervals allows for an optimized terminal time and mass of 3.248 and 0.755, respectively. However, with four intervals, the solution converged after 94 iterations and an elapsed

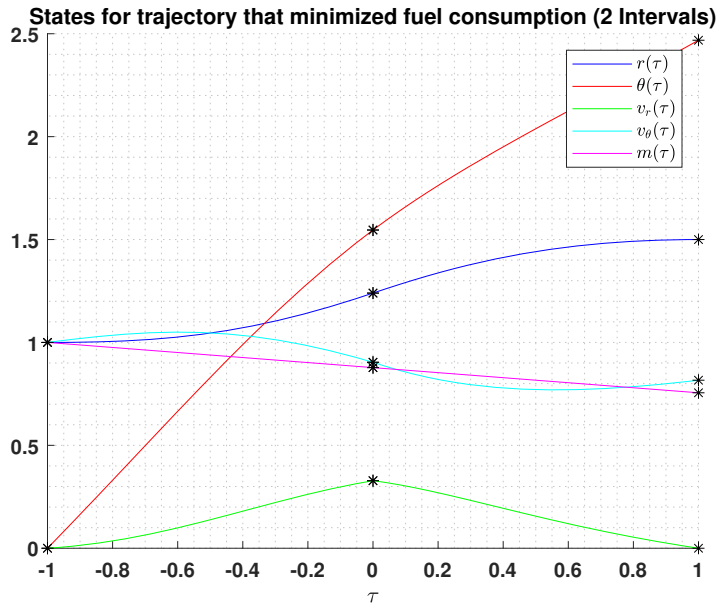

Figure 25: States for trajectory that minimized fuel consumption ($K : 2\ , N : 2$)

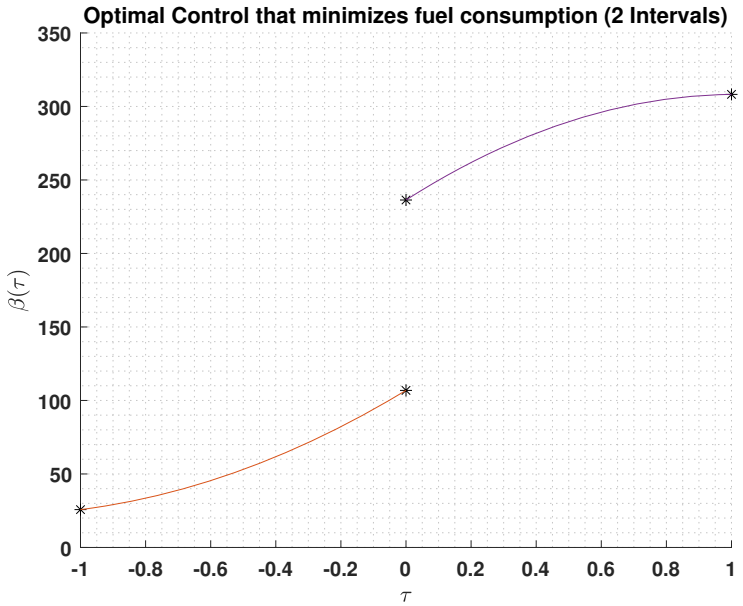

Figure 26: Optimal Control that minimized fuel consumption ($K : 2\ , N : 2$)

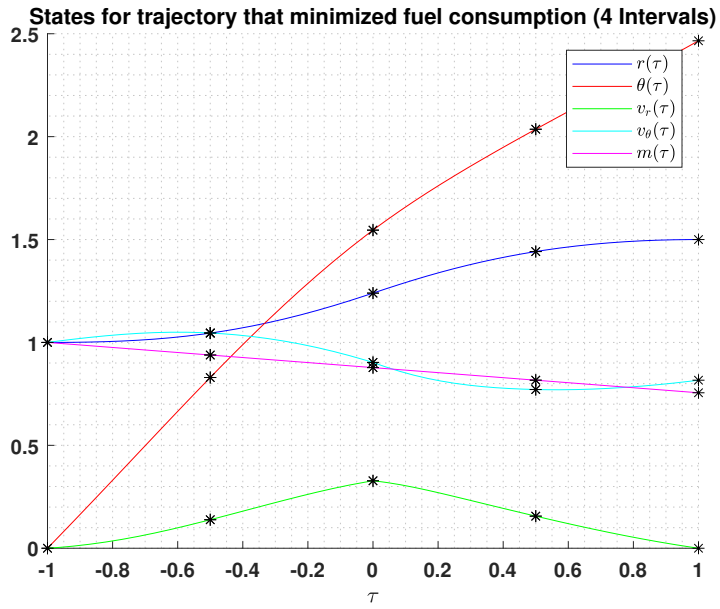

Figure 27: States for trajectory that minimized fuel consumption ($K : 4$, $N : 2$)

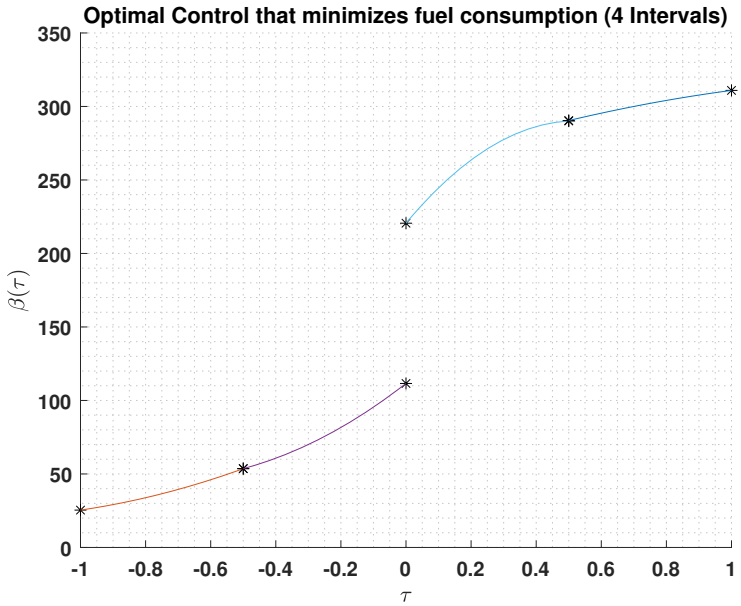

Figure 28: Optimal Control that minimized fuel consumption ($K : 4$, $N : 2$)

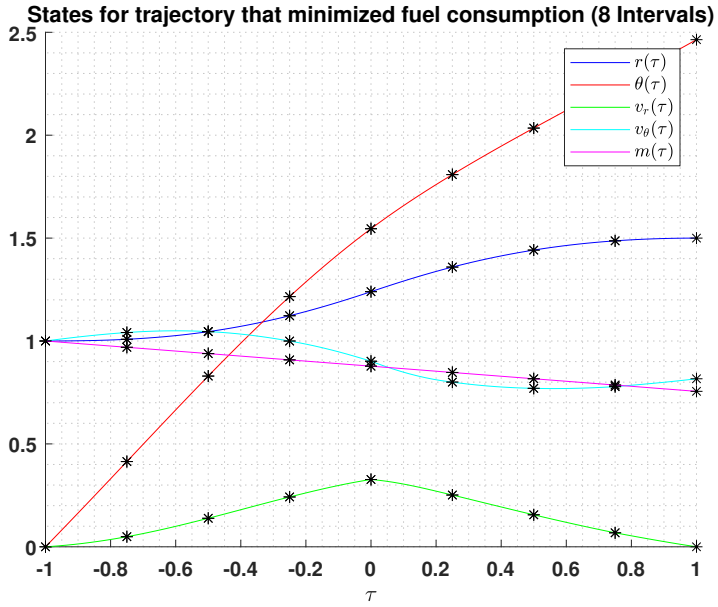

Figure 29: States for trajectory that minimized fuel consumption ($K : 8$, $N : 2$)

time of 18.411803 seconds. By this point, the addition of more intervals does not produce a more optimized solution while the computation cost continues to get larger. Figure 31 shows the states for the trajectory that minimized the fuel consumption for $K = 16$ and $N = 2$, while Figure 32 shows the optimal control to achieve this trajectory. The terminal time and mass were optimized to be 3.248 and 0.755, respectively. The solution converged after 112 iterations and an elapsed time of 80.721927 seconds. Interestingly about this case, a solution was originally computed with lower coefficient bounds which prevented the terminal conditions from being as optimal as previous cases. Once the bounds were opened, similar optimized terminal conditions were achieved.

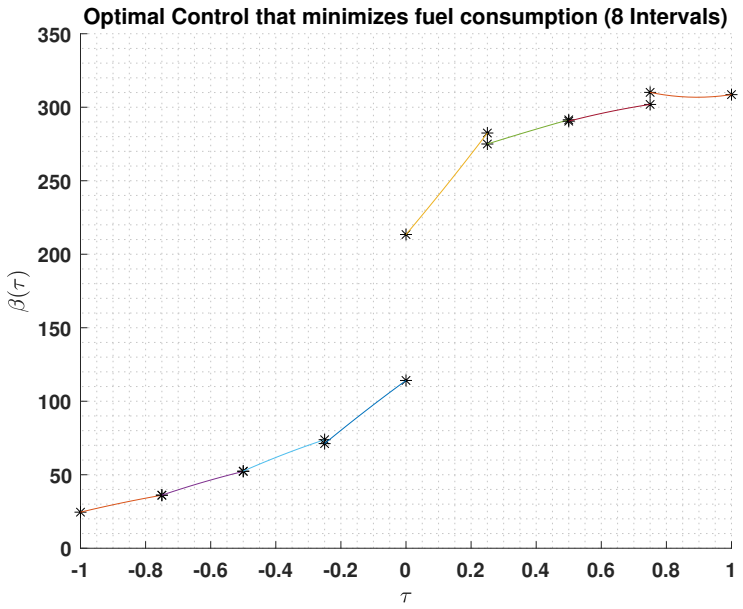

Figure 30: Optimal Control that minimized fuel consumption ($K : 8$, $N : 2$)

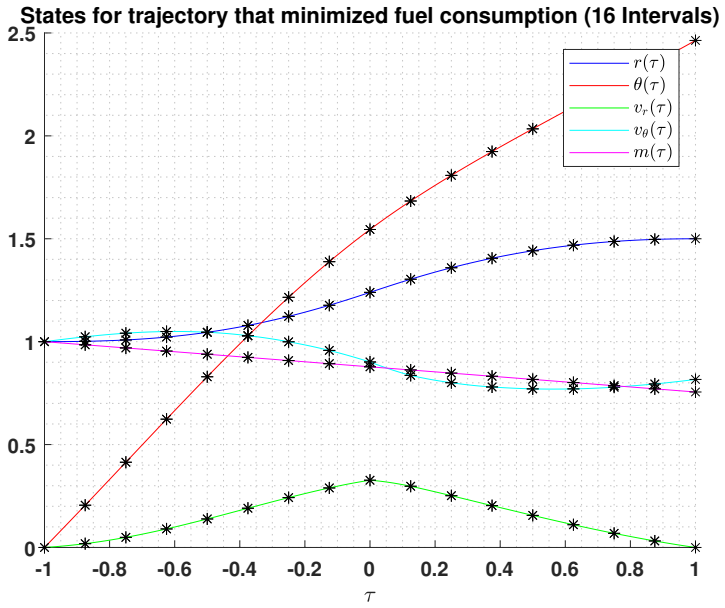

Figure 31: States for trajectory that minimized fuel consumption ($K : 16$, $N : 2$)

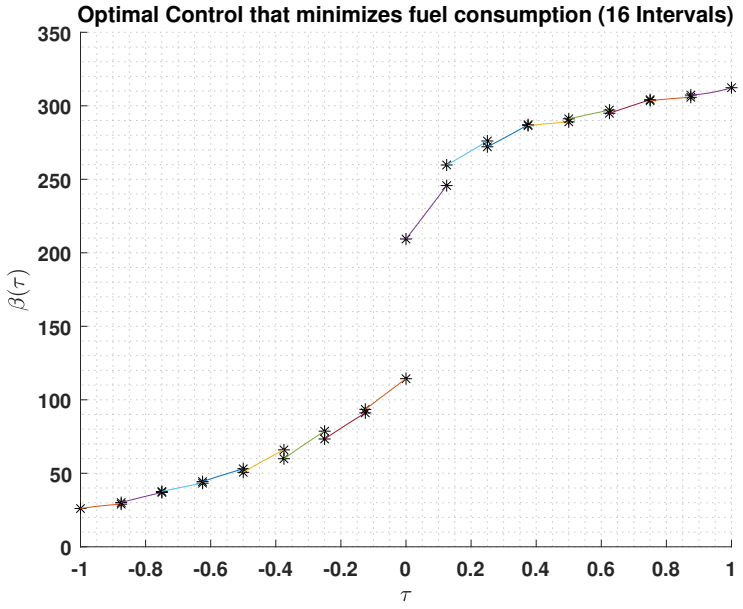

Figure 32: Optimal Control that minimized fuel consumption ($K : 16$, $N : 2$)

The next set of Direct Multiple Shooting cases are performed with the degree polynomial $N = 3$ and intervals $K = (2, 4, 8, 16)$. Figure 33 shows the states for the trajectory that minimized the fuel consumption for $K = 2$ and $N = 3$, while Figure 34 shows the optimal control to achieve this trajectory. The results for this case are similar to the previous Direct Multiple Shooting results. The terminal time was optimized to be 3.248 while the terminal mass was optimized to be 0.755. The solution for this case converged after 46 iterations and an elapsed time of 1.24360 seconds.

Figure 35 shows the states for the trajectory that minimized the fuel consumption for $K = 4$ and $N = 3$, while Figure 36 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The solution converged after 69 iterations and an elapsed time of 4.702343 seconds.

Figure 37 shows the states for the trajectory that minimized the fuel consumption for $K = 8$ and $N = 3$, while Figure 38 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The solution converged after 93 iterations and an elapsed time of 20.779223 seconds. Similar to Direct Multiple Shooting with polynomial degree $N = 2$, increase in intervals, does not improve the optimized solution and continues to increase in computation cost.

The last case where the control is parameterized by a 3^{rd} degree polynomial is with intervals $K = 16$. Figure 39 shows the states for the trajectory that

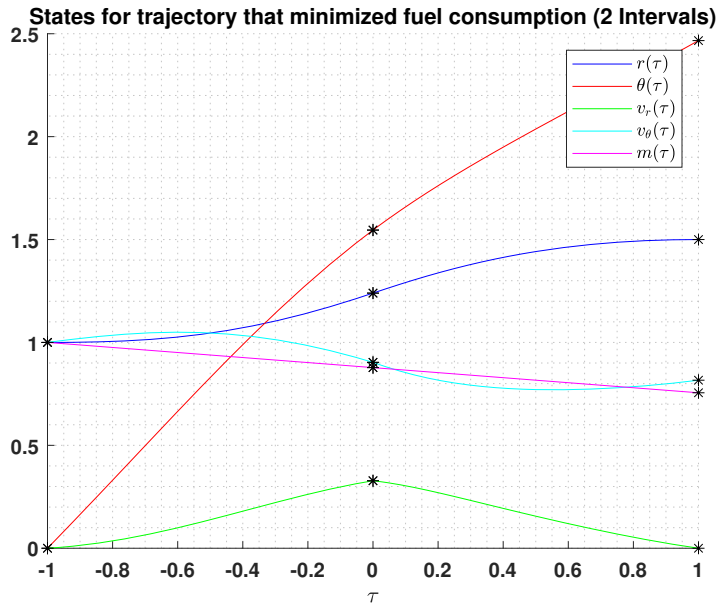

Figure 33: States for trajectory that minimized fuel consumption ($K : 2\ , N : 3$)

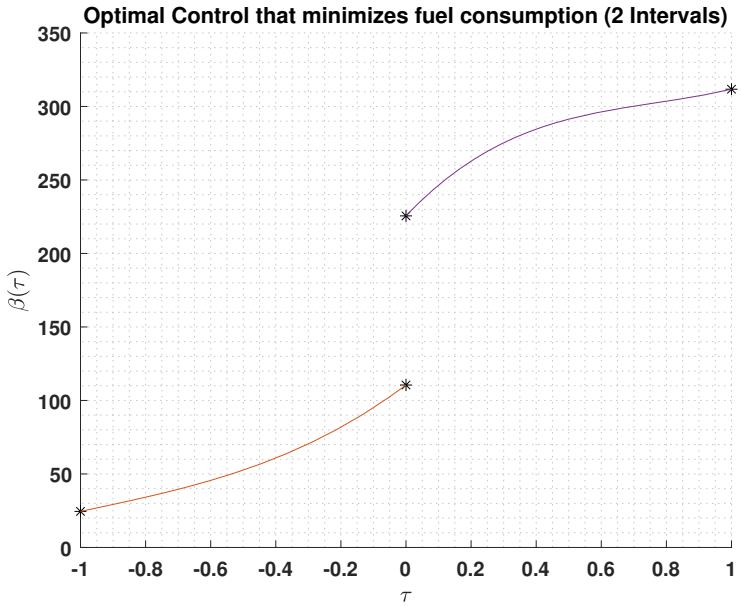

Figure 34: Optimal Control that minimized fuel consumption ($K : 2\ , N : 3$)

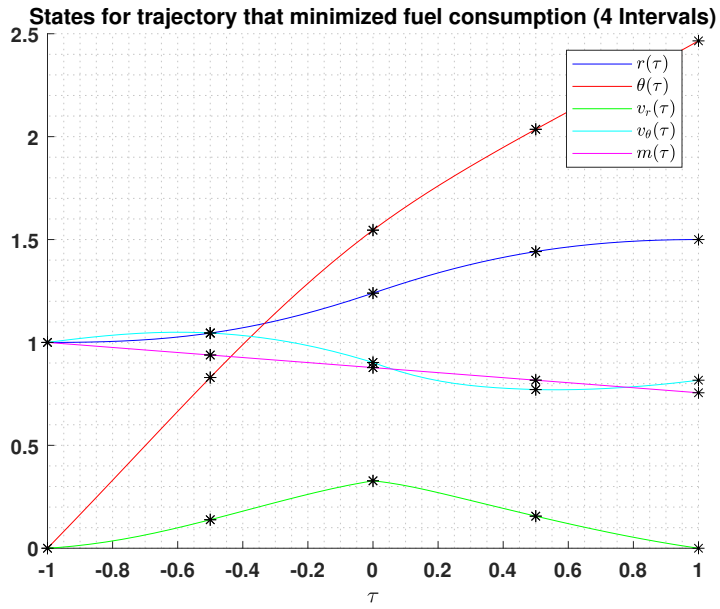

Figure 35: States for trajectory that minimized fuel consumption ($K : 4$, $N : 3$)

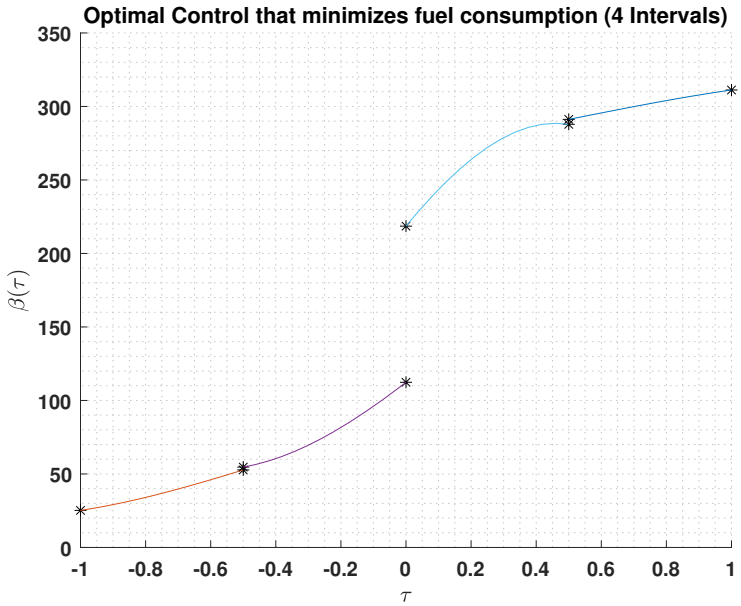

Figure 36: Optimal Control that minimized fuel consumption ($K : 4$, $N : 3$)

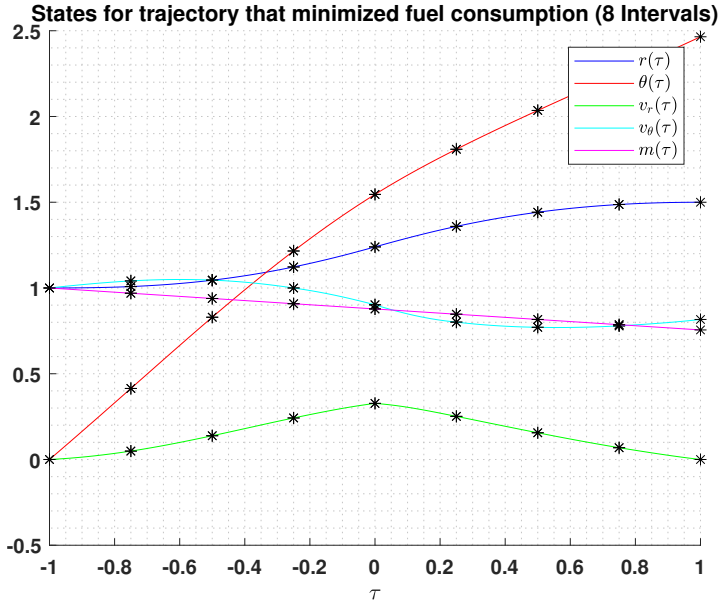

Figure 37: States for trajectory that minimized fuel consumption ($K : 8$, $N : 3$)

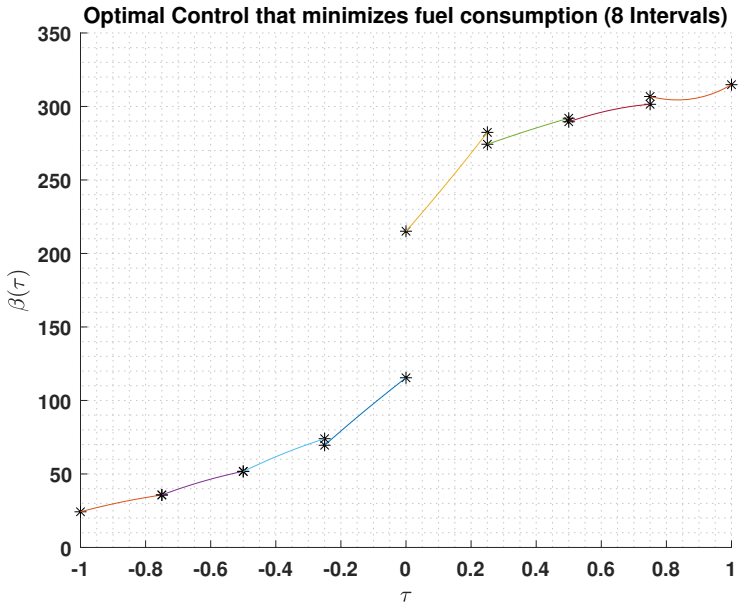

Figure 38: Optimal Control that minimized fuel consumption ($K : 8$, $N : 3$)

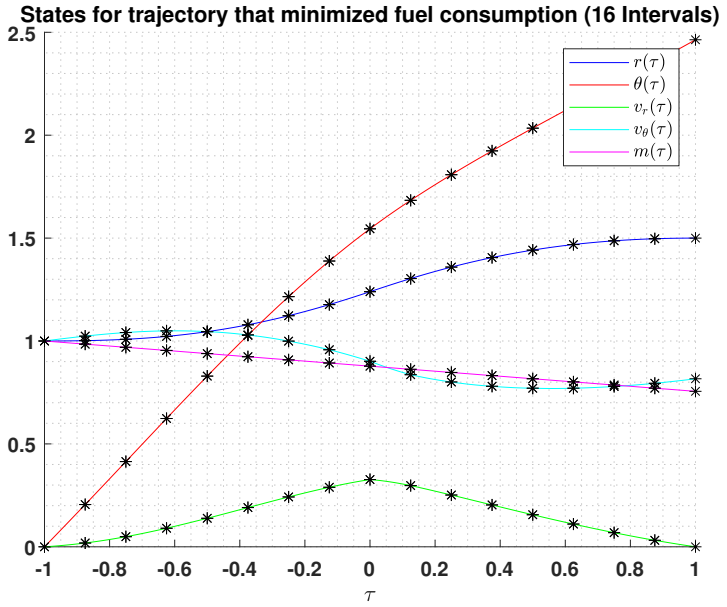

Figure 39: States for trajectory that minimized fuel consumption (K : 16 , N : 3)

minimized the fuel consumption, while Figure 40 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The solution converged after 128 iterations and an elapsed time of 105.069901 seconds. Just like 16 interval case with 2^{nd} degree polynomial control, the solution was further optimized when the polynomial coefficient bounds were opened. Increasing the bounds not only allowed for a more optimized solution, but also improved the computation cost.

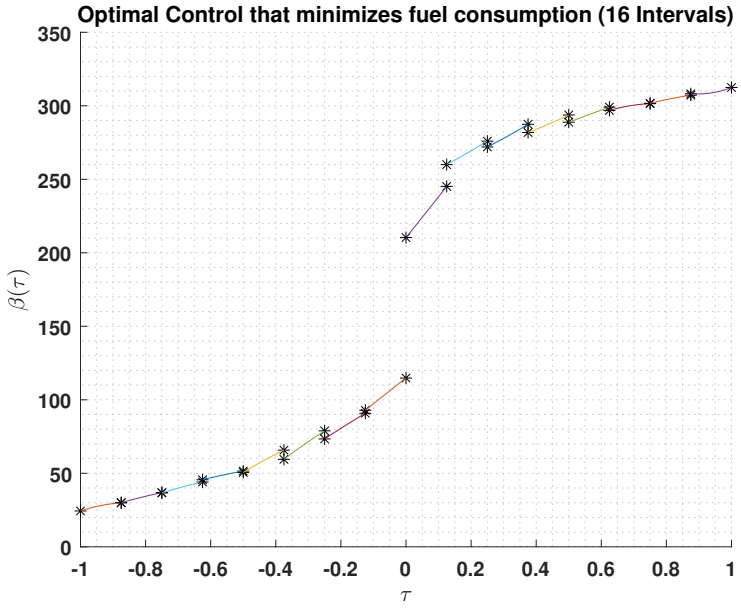

Figure 40: Optimal Control that minimized fuel consumption ($K : 16$, $N : 3$)

The third set of Direct Multiple Shooting cases are performed with the control parameterized by a 4^{th} degree polynomial for intervals $K = (2, 4, 8, 16)$. Figure 41 shows the states for the trajectory that minimized the fuel consumption for $K = 2$ and $N = 4$, while Figure 42 shows the optimal control to achieve this trajectory. The optimal control looks similar to that of the other direct multiple shooting two-interval cases. The optimized terminal time and mass are 3.248 and 0.7553, respectively. The optimal solution converged after 41 iterations and an elapsed time of 1.313428 seconds. Comparing with the other direct multiple shooting two-interval interval cases, it seems that the computation cost increases as the polynomial degree gets larger. This makes sense because an extra degree requires extra optimization.

Figure 43 shows the states for the trajectory that minimized the fuel consumption for $K = 4$ and $N = 4$, while Figure 44 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2479 and 0.7554, respectively. The optimal solution converged after 65 iterations and an elapsed time of 4.854514 seconds.

Figure 45 shows the states for the trajectory that minimized the fuel consumption for $K = 8$ and $N = 4$, while Figure 46 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2481 and 0.7554, respectively. The optimal solution converged after 90 iterations and an elapsed time of 22.244379 seconds.

Figure 47 shows the states for the trajectory that minimized the fuel consump-

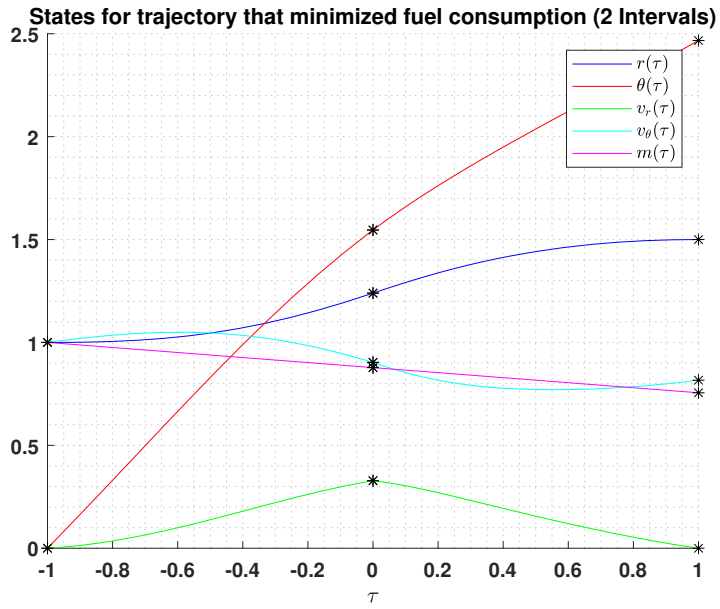

Figure 41: States for trajectory that minimized fuel consumption ($K : 2\ , N : 4$)

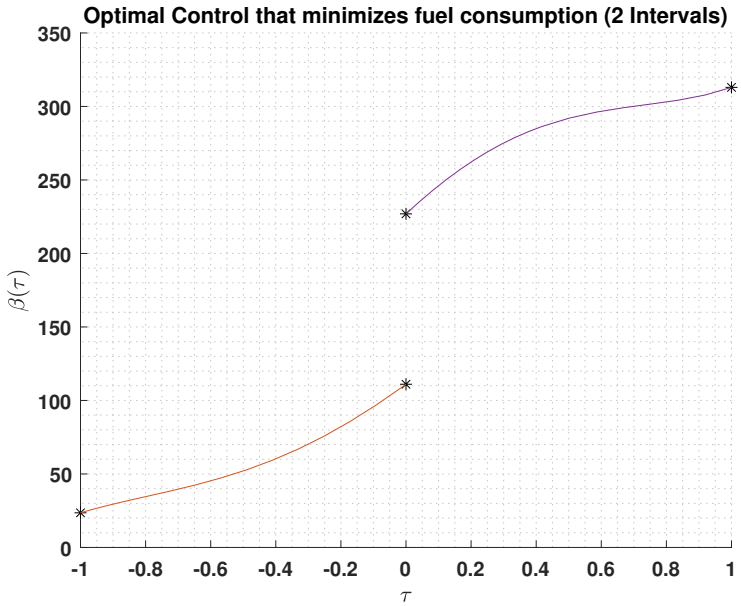

Figure 42: Optimal Control that minimized fuel consumption ($K : 2\ , N : 4$)

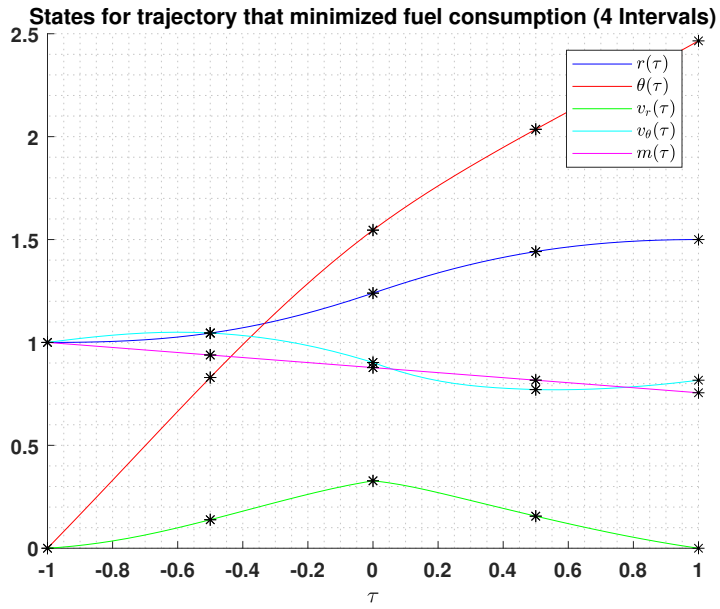

Figure 43: States for trajectory that minimized fuel consumption ($K : 4$, $N : 4$)

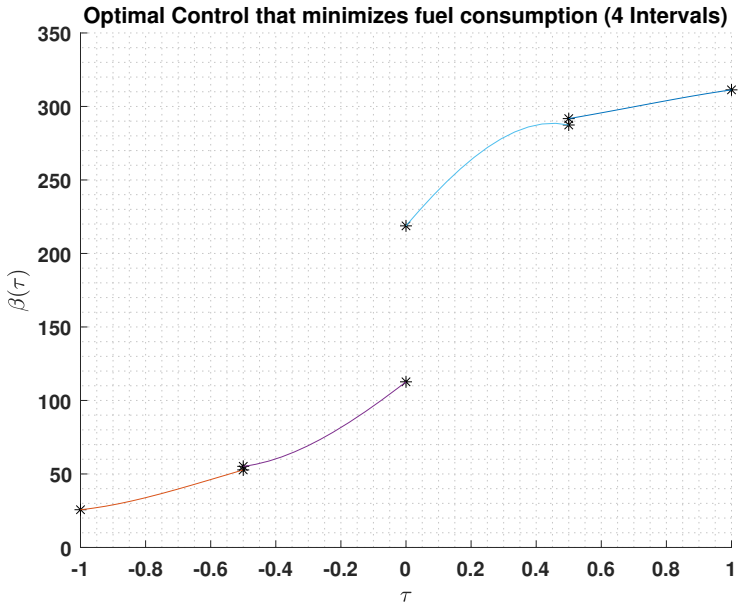

Figure 44: Optimal Control that minimized fuel consumption ($K : 4$, $N : 4$)

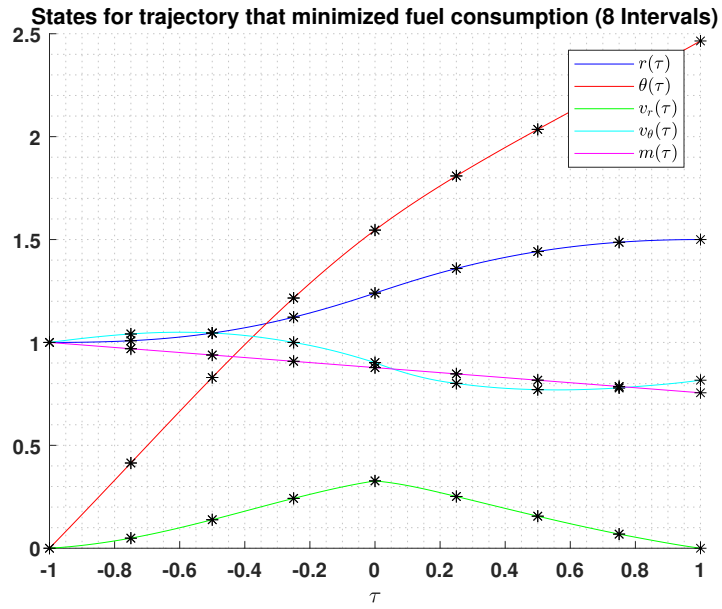

Figure 45: States for trajectory that minimized fuel consumption ($K : 8$, $N : 4$)

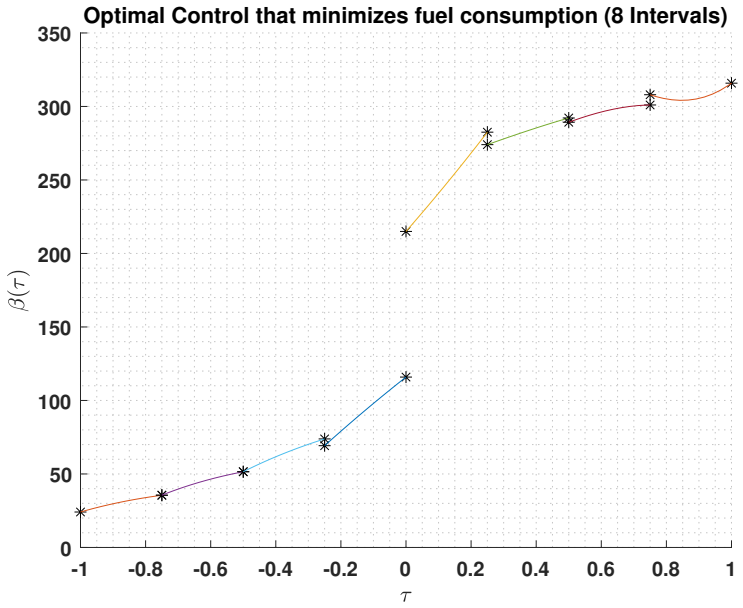

Figure 46: Optimal Control that minimized fuel consumption ($K : 8$, $N : 4$)

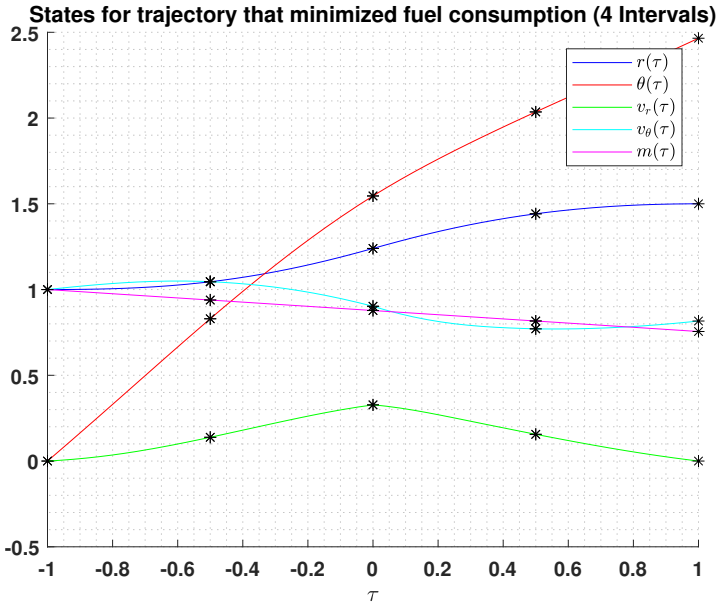

Figure 47: States for trajectory that minimized fuel consumption (K : 16 , N : 4)

tion for $K = 16$ and $N = 4$, while Figure 48 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2481 and 0.7554, respectively. The optimal solution converged after 142 iterations and an elapsed time of 129.659140 seconds. Again, like the other 16-interval cases, the polynomial coefficient bounds were opened for better optimization.

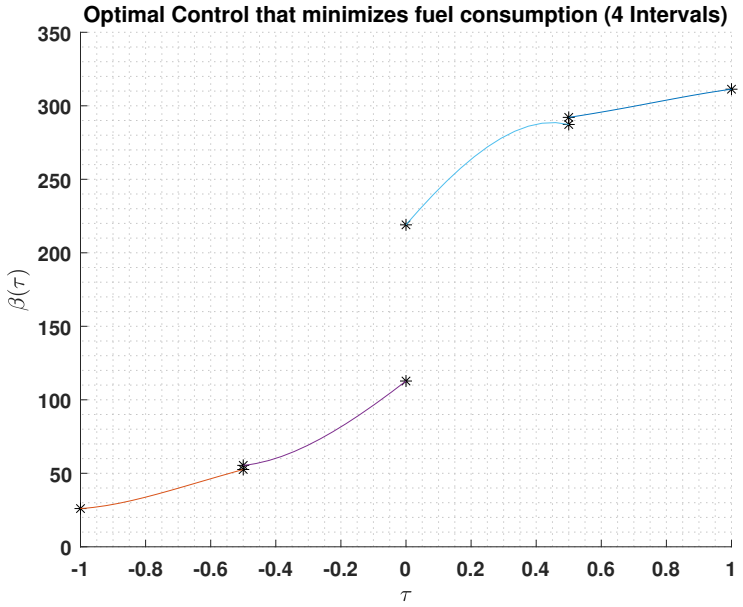

Figure 48: Optimal Control that minimized fuel consumption ($K : 16$, $N : 4$)

The next set of Direct Multiple Shooting cases are the control parameterized by a 5^{th} degree polynomial for intervals $K = (2, 4, 8, 16)$. Figure 49 shows the states for the trajectory that minimized the fuel consumption for $K = 2$ and $N = 5$, while Figure 50 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2485 and 0.7553, respectively. The optimal solution converged after 37 iterations and an elapsed time of 1.345495 seconds. Regardless of the polynomial degree, the optimized control seems to look very similar for each interval.

Figure 51 shows the states for the trajectory that minimized the fuel consumption for $K = 4$ and $N = 5$, while Figure 52 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2479 and 0.7554, respectively. The optimal solution converged after 67 iterations and an elapsed time of 5.625521 seconds.

Figure 53 shows the states for the trajectory that minimized the fuel consumption for $K = 8$ and $N = 5$, while Figure 54 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The optimal solution converged after 98 iterations and an elapsed time of 27.079276 seconds.

Figure 55 shows the states for the trajectory that minimized the fuel consumption for $K = 16$ and $N = 5$, while Figure 56 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The optimal solution converged after 136 iterations and

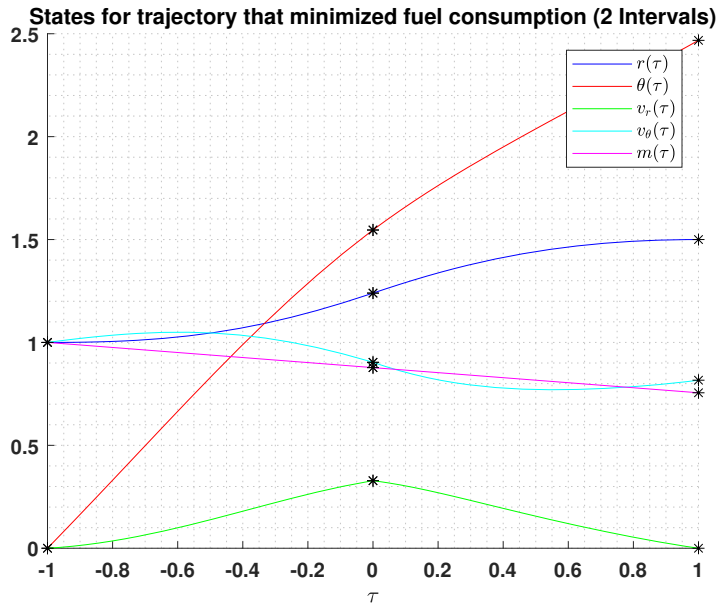

Figure 49: States for trajectory that minimized fuel consumption ($K : 2\ , N : 5$)

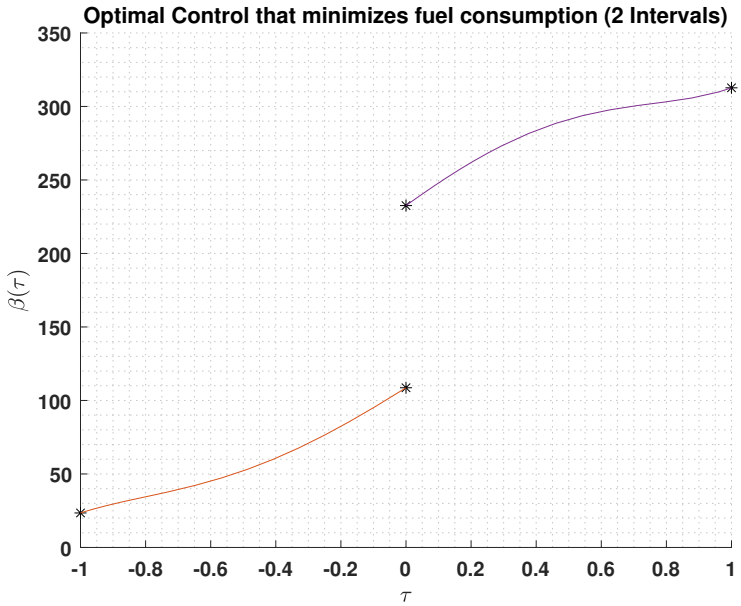

Figure 50: Optimal Control that minimized fuel consumption ($K : 2\ , N : 5$)

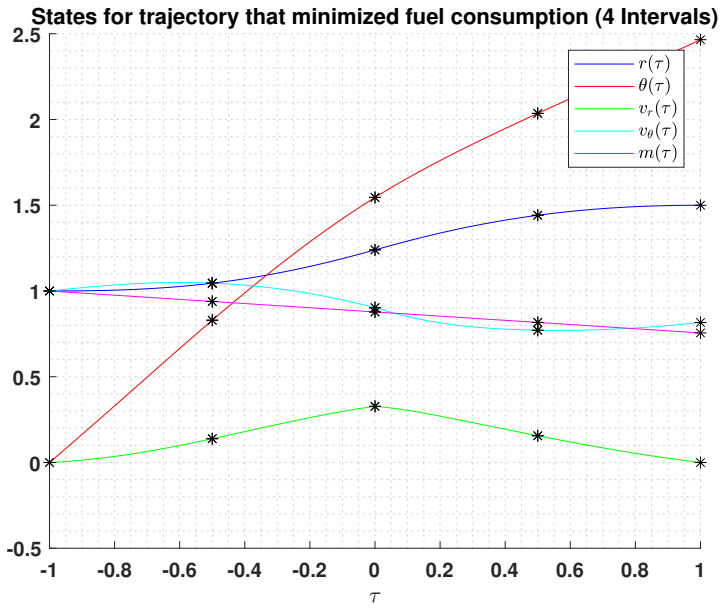

Figure 51: States for trajectory that minimized fuel consumption ($K : 4$, $N : 5$)

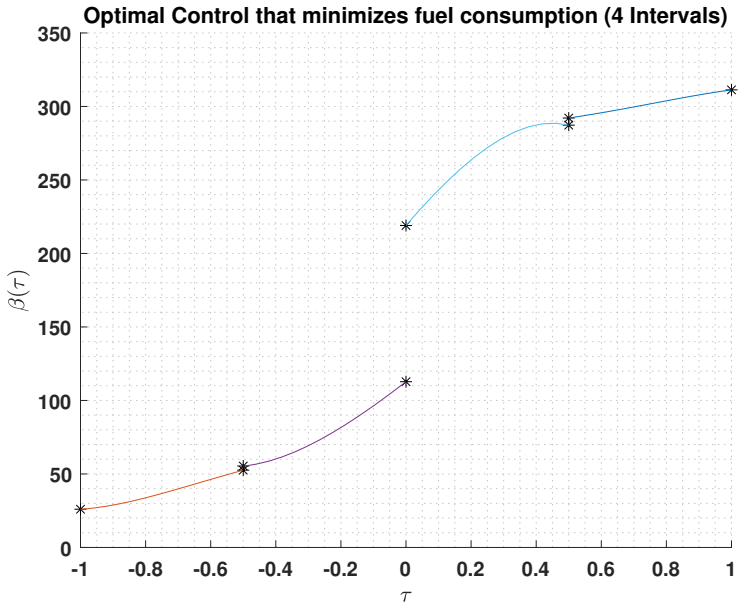

Figure 52: Optimal Control that minimized fuel consumption ($K : 4$, $N : 5$)

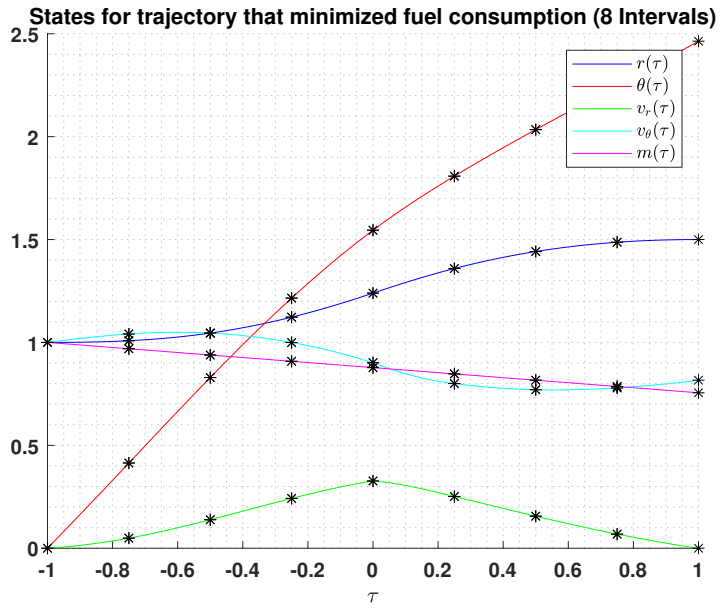

Figure 53: States for trajectory that minimized fuel consumption ($K : 8$, $N : 5$)

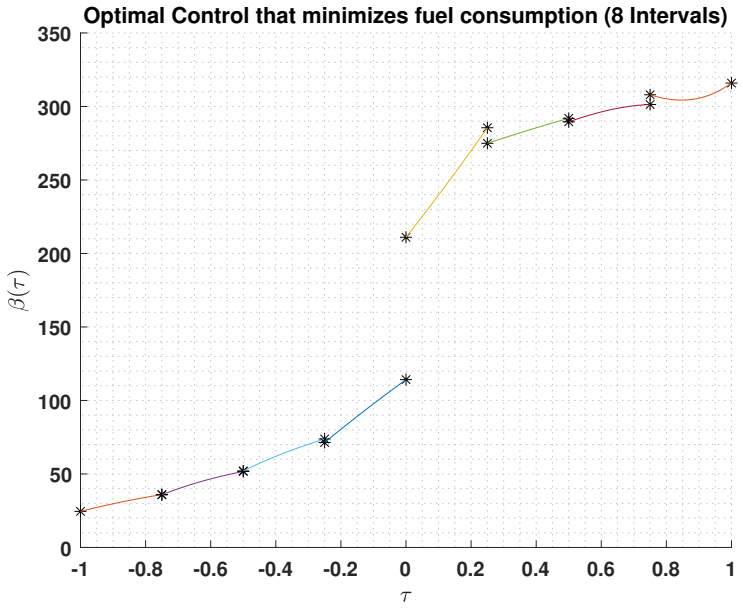

Figure 54: Optimal Control that minimized fuel consumption ($K : 8$, $N : 5$)

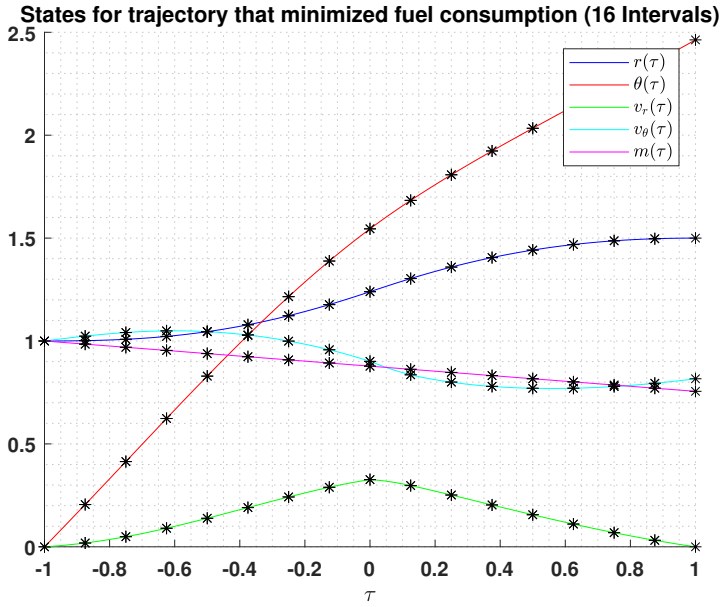

Figure 55: States for trajectory that minimized fuel consumption ($K : 16$, $N : 5$)

an elapsed time of 138.476447 seconds. Again, like the other 16-interval cases, the polynomial coefficient bounds were opened for better optimization. The terminal time changed from 3.25 to 3.248 which is a 0.06% difference.

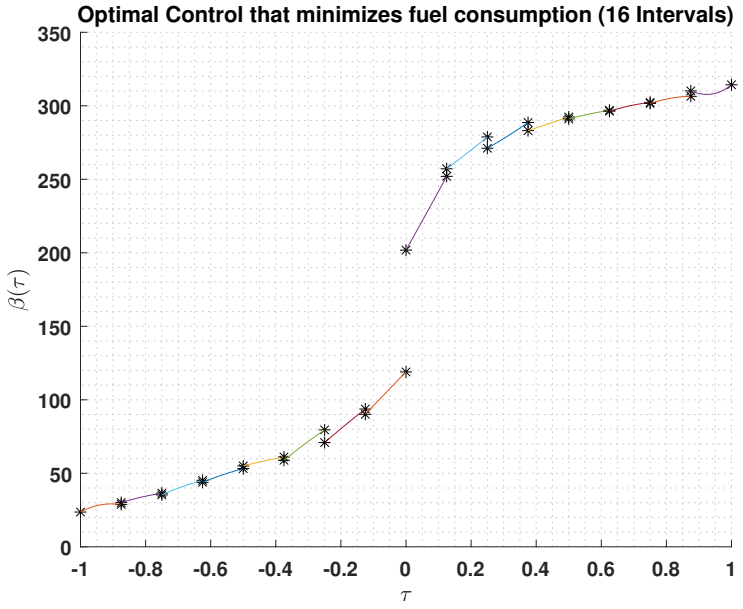

Figure 56: Optimal Control that minimized fuel consumption ($K : 16$, $N : 5$)

The last set of Direct Multiple Shooting cases are those with the control parameterized by a 6^{th} degree polynomial for intervals $K = (2, 4, 8, 16)$. Figure 57 shows the states for the trajectory that minimized the fuel consumption for $K = 2$ and $N = 6$, while Figure 58 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2486 and 0.7554, respectively. The optimal solution converged after 48 iterations and an elapsed time of 1.721967 seconds.

Figure 59 shows the states for the trajectory that minimized the fuel consumption for $K = 4$ and $N = 6$, while Figure 60 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.2479 and 0.755, respectively. The optimal solution converged after 63 iterations and an elapsed time of 5.7287548 seconds.

Figure 61 shows the states for the trajectory that minimized the fuel consumption for $K = 8$ and $N = 6$, while Figure 62 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The optimal solution converged after 107 iterations and an elapsed time of 31.923118 seconds.

The last case where the control is parameterized by a 6^{th} degree polynomial is with intervals $K = 16$. Figure 63 shows the states for the trajectory that minimized the fuel consumption, while Figure 64 shows the optimal control to achieve this trajectory. The optimized terminal time and mass are 3.248 and 0.7554, respectively. The solution converged after 128 iterations and an elapsed

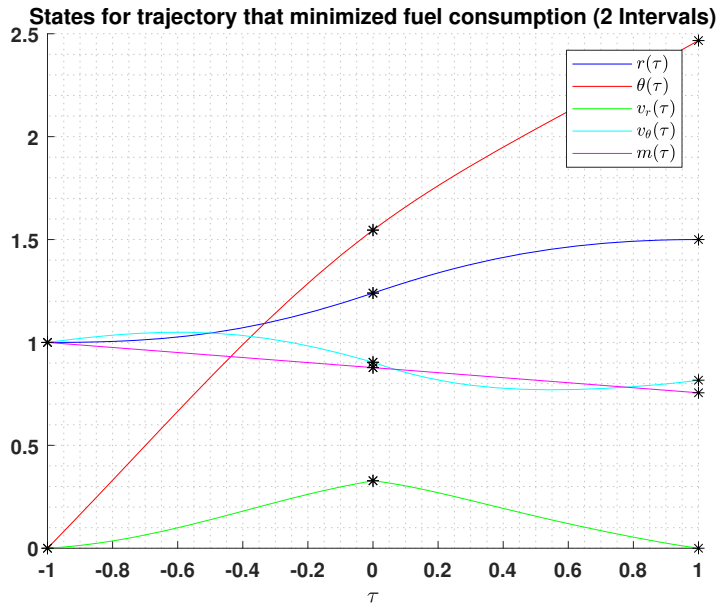

Figure 57: States for trajectory that minimized fuel consumption ($K : 2\ , N : 6$)

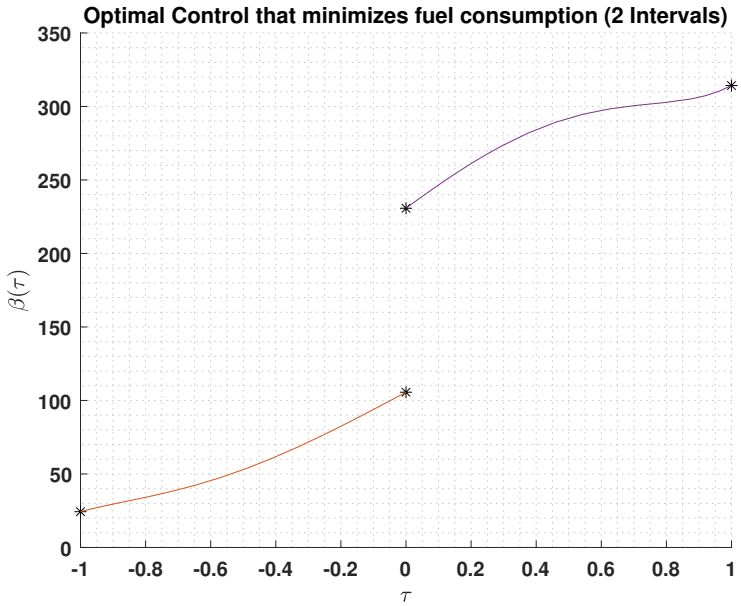

Figure 58: Optimal Control that minimized fuel consumption ($K : 2\ , N : 6$)

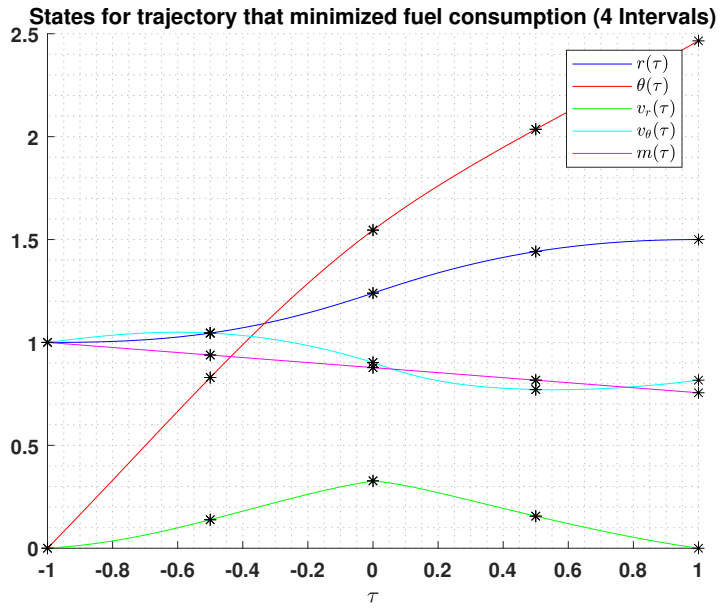

Figure 59: States for trajectory that minimized fuel consumption ($K : 4$, $N : 6$)

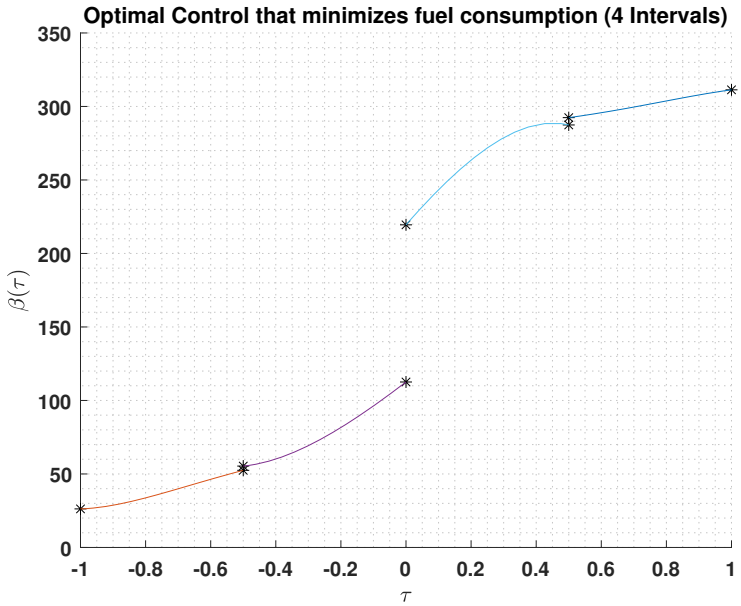

Figure 60: Optimal Control that minimized fuel consumption ($K : 4$, $N : 6$)

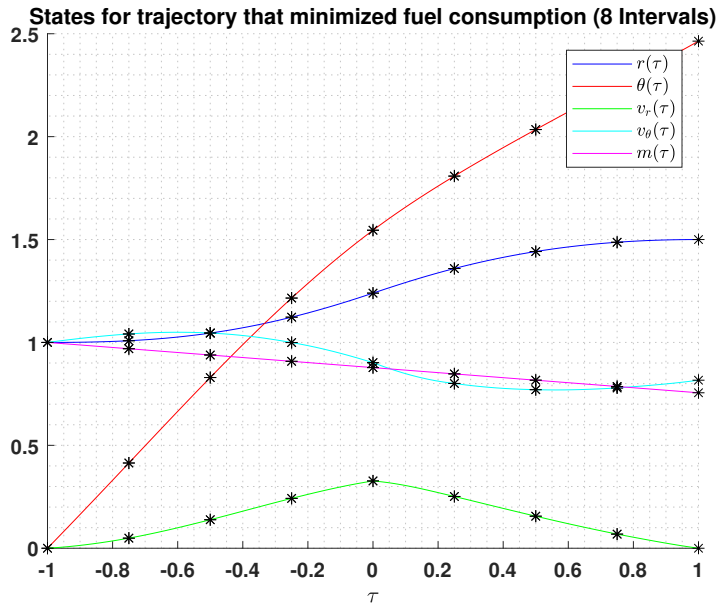

Figure 61: States for trajectory that minimized fuel consumption ($K : 8$, $N : 6$)

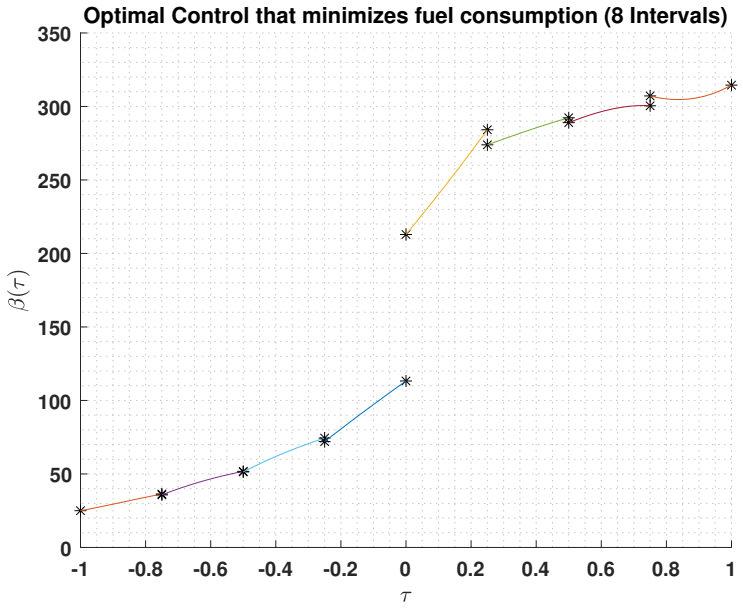

Figure 62: Optimal Control that minimized fuel consumption ($K : 8$, $N : 6$)

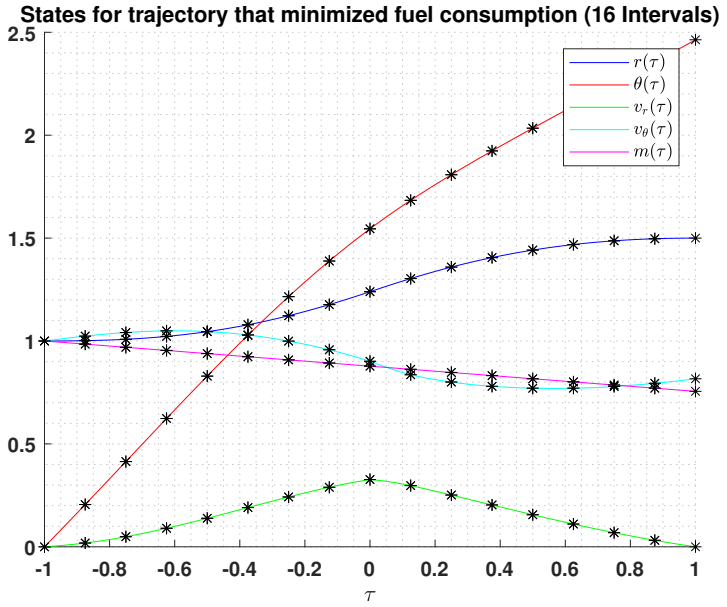

Figure 63: States for trajectory that minimized fuel consumption ($K : 16$, $N : 6$)

time of 184.219058 seconds. Just like 16 interval cases with N degree polynomial control, the solution was further optimized when the polynomial coefficient bounds were opened. Increasing the bounds not only allowed for a more optimized solution, but also improved the computation cost.

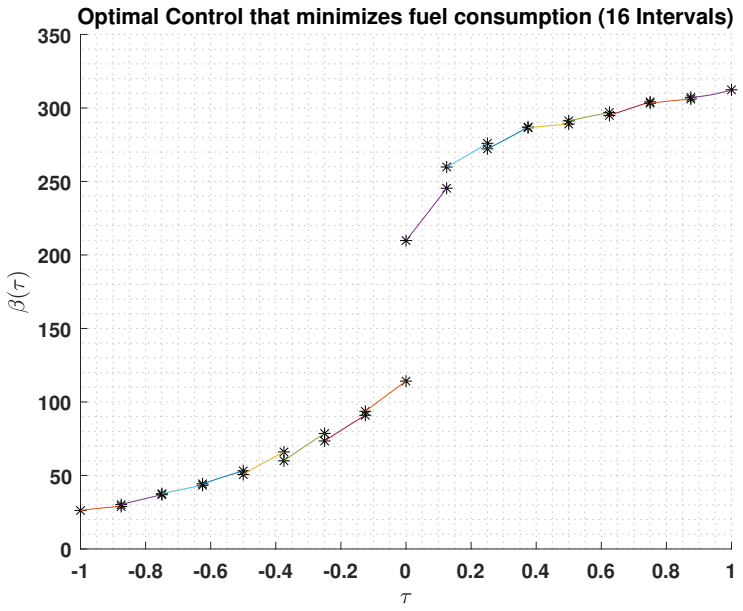

Figure 64: Optimal Control that minimized fuel consumption ($K : 16$, $N : 6$)

Degree-N	Intervals-K	Iterations (s)	Sim Time	Terminal Time	Terminal Mass
2	2	33	0.7972	3.24888	0.75535
2	4	59	3.0869	3.24785	0.75543
2	8	75	13.8941	3.24780	0.75544
2	16	112	79.1192	3.24771	0.75544
3	2	45	0.8537	3.24804	0.75542
3	4	63	3.5958	3.24786	0.75543
3	8	92	19.576	3.24784	0.75543
3	16	128	101.136	3.24772	0.75544
4	2	38	0.7176	3.24829	0.75540
4	4	66	4.1511	3.24792	0.75543
4	8	87	20.2003	3.24793	0.75543
4	16	142	124.693	3.24750	0.75555
5	2	38	0.8166	3.24852	0.75538
5	4	65	4.7263	3.24795	0.75542
5	8	94	24.1589	3.24783	0.75543
5	16	136	132.165	3.24759	0.75545
6	2	47	1.1136	3.24863	0.75537
6	4	64	4.9823	3.24796	0.75542
6	8	117	35.1663	3.24778	0.75544
6	16	145	153.775	3.24760	0.75545

Table 4: Performance for Direct Multiple Shooting

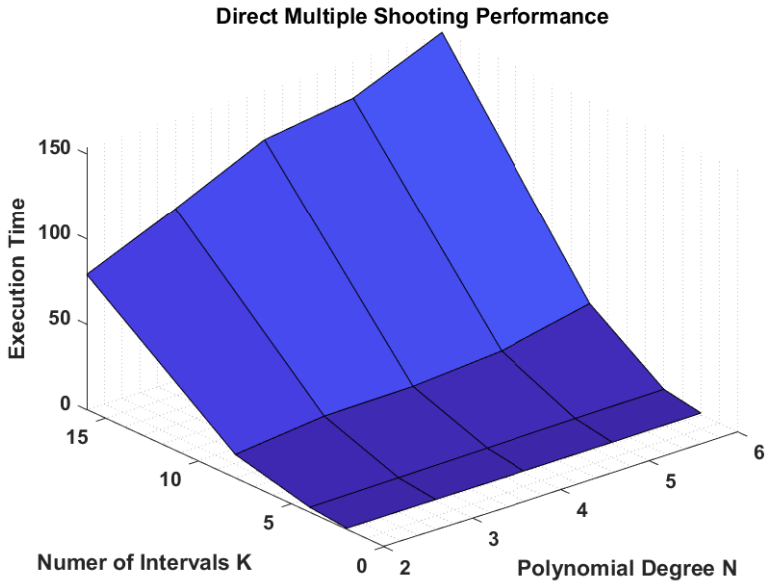

Figure 65: Statistics for all Direct Multiple Shooting Cases

Figure 66: Execution Times for all Direct Multiple Shoot Cases

3.4.1 Analysis of Direct Multiple Shooting

The Direct Multiple Shooting numerical method produced much better results than Direct Shooting. While Direct Shooting had trouble optimizing the control, Direct Multiple Shooting was able to optimize the control almost as good as the indirect methods. The difference between the two direct methods is that Direct Multiple Shooting allows for the use of multiple, rather than single polynomials. When looking at the results in Table 4, it is noticeable that the degree of the polynomial has negligible impact on converging to the optimal solution. Examining Figure 67, it is obvious that the polynomial degree has a slight affect on the terminal time but unless a 0.003% difference is a deal-breaker, then it can be considered negligible. The main difference is that increasing the polynomial degree also increases the computation cost. For example, the solution for the case of 2 intervals with polynomial degree 2 converged after 33 iterations and 0.7972 seconds while the solution for the case of 2 intervals with polynomial degree 6 converged after 47 iterations and 1.1136 seconds. This is approximately a 42.3% increase in computation time. While this percentage may seem high, the magnitude of the difference for this example is low. Because of this, for some applications, the extra computation time may not be significant. Meanwhile, the 16-interval case amplified the loss in efficiency. For the case of 16 intervals with polynomial degree 2, the solution converged in 79.1 seconds while the solution for the case of 16 intervals with polynomial degree 6 converged in 153.8 seconds. This is approximately a 74 second difference which is about a 94.4% increase. Figure 66 shows how the polynomial degree increase has a much greater impact as the number of intervals also increases. Interesting, you can almost fit a quadratic to the change in time as intervals increase. Figure 65 is another good representation of how the polynomial degree and number of intervals affects the execution time. This phenomena is significant and shows that for this particular problem, the optimal control is more efficiently parameterized by a 2^{nd} degree polynomial rather than a 6^{th} degree polynomial. If Direct Multiple Shooting was the only option, Figure 67 would be a good source to help make design decisions. Depending on execution constraints and desired optimization, a polynomial and interval combination can easily be selected from the figure. For example, if the cost of minimizing the objective function was much higher than the cost if computation, then polynomial degree 6 with 16 intervals is the optimal choice. On the other hand, if the cost of computation is high and the cost of minimizing the objective function was low, then 2 intervals with polynomial degree 2 is the optimal choice.

3.5 Overall Analysis

For this particular problem, the solution was best achieved using the Indirect Shooting Method. As seen in Table 1, the objective function was minimized, minimize terminal time or fuel burned, in only 0.156 seconds. The final time was found to be 3.247 which is the lowest that any case was able to achieve. The terminal mass was found to be 0.755 which is the most that any case achieved.

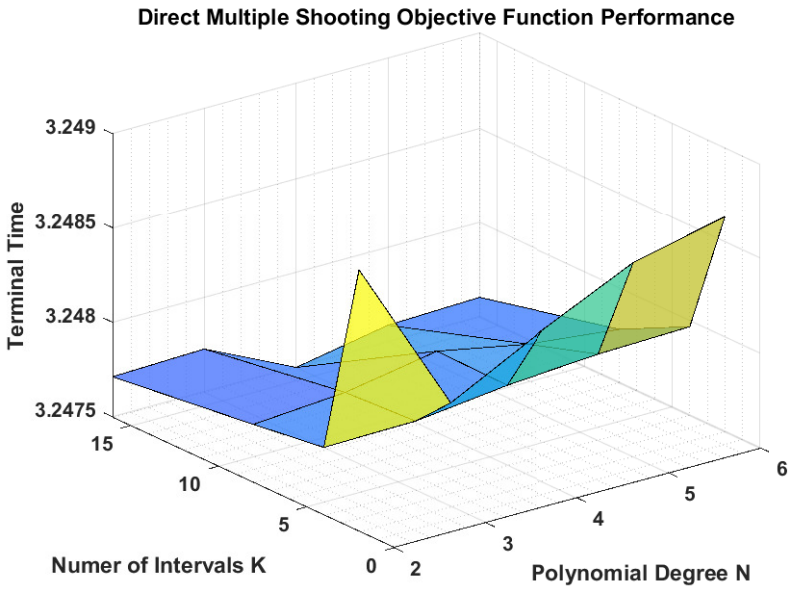

Figure 67: Minimized Time for all Direct Multiple Shooting Cases

Indirect Shooting also had a continuous optimal control which is the most realistic, since the control is the angle of the thrust force. Because of this, Indirect Shooting will be used as a baseline for the remaining analysis. The objective function obtained by the Indirect Mutliple Shooting Method was also well optimized. Each case, intervals $K = (2, 4, 8, 16)$, was able to achieve the same terminal time and mass as the Indirect Shooting Method. However, if you look at Table 2, you'll realize that the sim-time was longer for each case compared to that of Indirect Shooting. This is due to optimization being performed at multiple intervals rather than just at the boundary point. For each additional interval, additional states and co-states errors must be minimized. Since the sim-time is greater and performance is equal for each Indirect Multiple Shooting case, it is more efficient to use Indirect Shooting for this optimal control problem. However, for many other problems, such as the hypersensitive problem, Indirect Multiple Shooting is a much better choice. Multiple Shooting is most effective when integration times are higher and not the objective function being minimized. This is because the solutions to many differential equations are exponential functions and when the time grows too large, modern computers cannot handle the precision. Because of this optimization may fail. For example, if the objective function for this problem would have been to follow a trajectory that required a large final time, then Indirect Shooting would most likely of failed. In this case, Indirect Multiple Shooting would have been the optimal method since it breaks the time interval into smaller sub-intervals. The

next numerical method used to solve the optimal control problem was Direct Shooting. Direct Shooting was performed with the control parameterized by a polynomial basis function for cases of polynomial degree $N = (2, 3, 4, 5, 6)$. As seen in Table 3, Direct Shooting was not able to minimize the objective function as well as the indirect methods. The best results, using Direct Shooting, were with the control parameterized by a 6^{th} degree polynomial. The terminal time and mass for this case was 3.3611 and 0.7469, respectively. This is a 3.5% increase in time and 1% increase in mass. The difference in performance between Direct Shooting and the indirect methods is due to how the control is being parameterized. For the indirect methods, the control is directly solved for with the Hamiltonian. For direct methods, the control is parameterized by basis functions such as polynomials. For this particular problem, the basis functions are polynomials with unknown coefficients. Because of this, the coefficients must be optimized alongside the states. While this control parameratization technique isn't necessarily worse than that of the indirect methods, Direct Shooting only used a single polynomial for the entire control, making it hard for the control to be optimized. If you compare Figure 6 with Figures 16, 18, 20, 22 and 24, it is obvious that a single polynomial doesn't optimize the control to the fullest potential. While the optimal solution takes a slight hit, Direct Shooting is very efficient. Direct Shooting with polynomial degree $N = 2$, had an execution time of 0.1380, which is faster than any other methods. Of course, the increase in efficiency results in a decrease in solution accuracy. For this case, the terminal time was 3.8662, which is approximately a 19% increase from the Indirect Shooting case. The maximum execution time for this method was the 6^{th} degree polynomial case which had an execution time of about 0.6769 seconds. While this is still higher than the Indirect Shooting case, it is lower than most of the Indirect Multiple Shooting cases. The increase in efficiency is due to the decrease in complexity of the control parameterization. Direct methods do not employ calculus of variations which in many cases, requires heavy computations. Since the control for this problem could be directly solved with the Hamiltonian and co-state equations, rather than requiring root-finding, the calculus of variations added minimal computation cost. Depending on the application, the efficiency could be major factor in the decision on what method to use. The final numerical method used to solve the optimal control problem was Direct Multiple Shooting. Twenty different cases were studied, which included all combinations of intervals $K = (2, 4, 8, 16)$, and polynomial degrees $N = (2, 3, 4, 5, 6)$. Unlike Direct Shooting, all combinations of Direct Multiple Shooting were able to minimize the objective function to an amount close to the baseline defined by Indirect Shooting. Figure 69 shows a comparison of every case's objective function (terminal mass). Although it's difficult to tell, but the Indirect, Indirect Multiple, and Direct Multiple cases lie on top of one another (top maroon line). All of the lines below the maroon line are Direct Shooting cases. If we take Direct Shooting out of the picture, Figure 70 shows a better comparison of the other methods. In this figure, both indirect methods lie on one another at the top. The rest of the lines indicate Direct Multiple Shooting cases. This is interesting because it shows that both indirect methods are able

to come up with a more optimal solution than the direct methods. This difference is most likely due to the control parameterization, as mentioned before. It is noticeable from Table 4 and Figure 70 that as you increase the number of intervals for Direct Multiple Shooting, the solution gets closer to the optimal solution achieved by Indirect and Indirect Multiple Shooting. This is because the increase in intervals allows for polynomials to better fit the optimal control. Also, in the current study, there were no constraints on the control being continuous. If continuity was enforced for Direct Multiple Shooting, then the optimal solution may have been achieved at the expense of computation time. For this particular problem, Direct Multiple Shooting was determined to be less efficient than any of the other methods. Indirect and Direct Shooting were both very efficient and had execution times less than 1 second long. Figure 68 shows the execution performance for all of the Indirect Multiple and Direct Multiple cases. It is clear that, regardless of the polynomial degree, Indirect Multiple Shooting has much better execution performance. For both methods, as the number of intervals increase, so does the execution time. Interestingly, the growth is much different between the two methods. For Indirect Multiple Shooting, the increase in execution time is almost linear to the increase in intervals. However, for Direct Multiple Shooting, the growth in execution time can almost be fit with a quadratic. This may be due to the fact that Indirect Multiple Shooting has more conditions (transversality) that help the errors converge quicker while Direct Multiple Shooting only has the boundary conditions. The performance for Indirect Multiple shooting may have been worse if the control couldn't be directly computed by the Hamiltonian and co-state equations. For instance, if a root-finding method was needed for the Indirect control, then computation times may have been much higher.

After performing an in-depth study, it is clear that Indirect Shooting is the numerical method that solved the current optimal control problem the best. While Indirect Shooting is the best method for this problem, it may not be optimal for many other problems. Indirect Shooting is not robust to large integration times and may fail when running on modern hardware. Indirect Multiple Shooting is much more robust since it breaks the time into smaller sub-intervals. A drawback to both indirect methods is that they require calculus of variations. For many applications, calculus of variations is very difficult to do, which is why, when done, it produces very accurate solutions. For cases where calculus of variations is too difficult to perform, or accuracy is not critical, Direct Multiple Shooting would be an optimal choice. The construction of direct methods is much more simple than indirect methods.

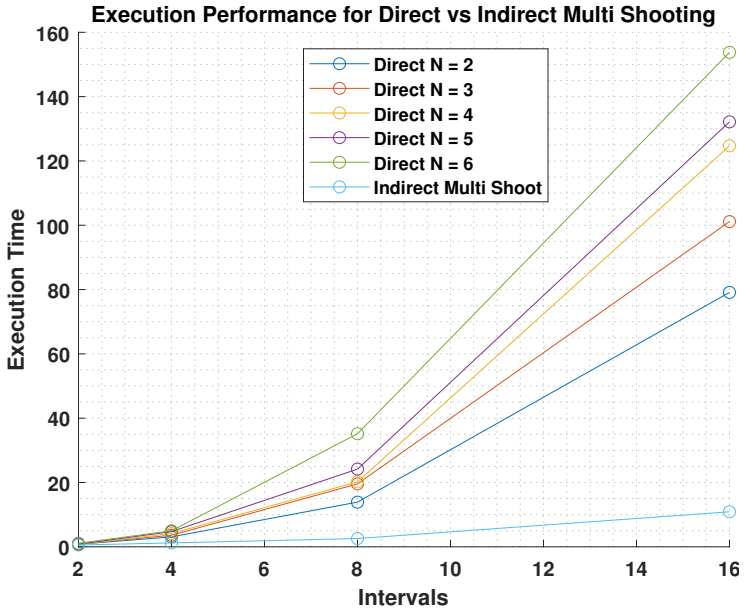

Figure 68: Direct Multi vs Indirect Multi Execution Performance

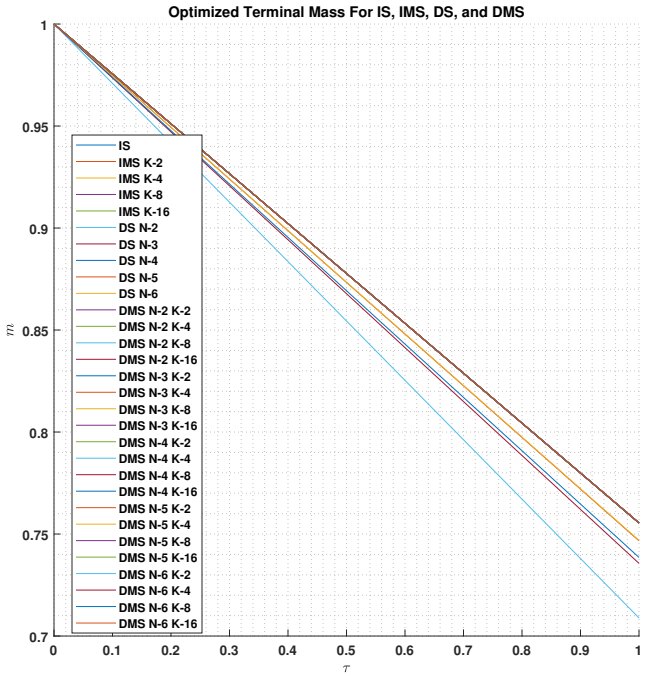

Figure 69: Terminal Mass Comparison of all Methods

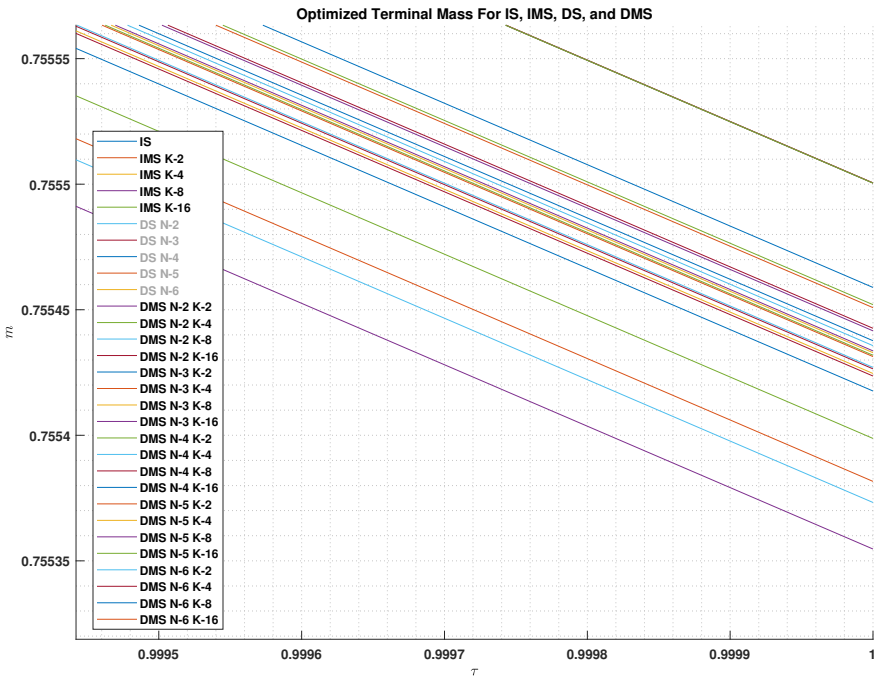

Figure 70: Terminal Mass Comparison of Indirect Methods vs Direct Multiple

4 Future Work

I would like to continue this study by solving the optimal control problem with other numerical methods such as Indirect and Direct Collocation. It would be interesting to compare these methods with the others. Also, I would like to change the problem to make it such that Indirect Shooting doesn't outperform the other methods. In reality, an orbit transfer problem would most likely not be solvable with simple indirect shooting and would require something more complicated. Lastly, I would like to re-visit the direct methods and put a constraint on the control to force continuity. This would make the control more realistic since a single thruster must be continuous.

5 Appendix

Listing 1: midterm_indirect_shooting_main.m

```
%% EML6934 Optimal Control
% Name: Elias Reyes
% Date: 04 April 2022
% Assignment: Midterm
% Goal: Solve the orbit transfer problem using indirect shooting
close all; clear all; clc
format longg
format compact

% given constants
param.T = 0.1405;
param.mew = 1;
param.ve = 1.8658344;
% boundary conditions
param.r0 = 1;
param.theta0 = 0;
param.vr0 = 0;
param.vtheta0 = sqrt(param.mew/param.r0);
param.rf = 1.5;
param.vrf = 0;
param.vthetaf = sqrt(param.mew/param.rf);
param.t0 = 0;
param.m0 = 1;
% transversality coniditions
param.lamthetaf = 0;
param.lammf = 1;
param.Hf = 0;
% initial guesses for unkown parameters
lamr = -2;
lamtheta = 0;
lamvtheta = 2;
lamvr = 2;
lamm = 2;
tf = 3.5;
zguess = [lamr; lamtheta; lamvtheta; lamvr; lamm; tf];

options = optimoptions('fsolve','Display','Iter','TolFun',1e-8);

% solve the optimal control problem
f = @(x)indirectOrbitTransferError(x,param);
tic
solution =fsolve(f, zguess ,options);
toc

```

```
%% Take optimal initial conditions and integrate them for plotting purposes
lam0 = solution(1:end-1); % optimal lambda values as t0
tf = solution(end); % optimal final time
X0 = [param.r0; param.theta0; param.vr0; param.vtheta0; param.m0; lam0];

options = odeset('reltol',1e-6);
tspan = [param.t0 tf];
tspan = [-1 1];
% integrate with optimal conditions
[t,p] = ode113(@orbitTransferOde,tspan,X0,options,param,tf);

lamvrf = p(:,8);
lamvthetaf = p(:,9);

% optimal control, use "mod" to ensure 0 -> 2pi
beta = mod(atan2(lamvrf,lamvthetaf),2*pi);

% plot figures
figure; hold on; grid minor
plot(t,p(:,1));
plot(t,p(:,2));
plot(t,p(:,3));
plot(t,p(:,4));
plot(t,p(:,5));
xlabel('t','Interpreter','LaTeX')
legend('r(t)','$\theta(t)$','vr(t)','v$\theta(t)$','m(t)','Interpreter','LaTeX')
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10)
title('States for trajectory that minimized fuel consumption')

figure; hold on; grid minor;
plot(t,beta*180/pi)
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10)
xlabel('t','Interpreter','LaTeX')
ylabel('$\beta(t)$','Interpreter','LaTeX')
title('Optimal Control that minimizes fuel consumption')
```

Listing 2: indirectOrbitTransferError.m

```
function E = indirectOrbitTransferError(z,param)
% error function for indirect shooting

lam0 = z(1:end-1);
tf = z(end);

% initial conditons for ode. uses boundary conditions and unkowns co-states
X0 = [param.r0; param.theta0; param.vr0; param.vtheta0; param.m0; lam0];

options = odeset('reltol',1e-6);
tspan = [param.t0 tf];
tspan = [-1 1];
% solve for states and co-states
[t,p] = ode113(@orbitTransferOde,tspan,X0,options,param,tf);

s = length(t);

% propagated states at tf
rf = p(s,1);
thetaf = p(s,2);
vrf = p(s,3);
vthetaf = p(s,4);
mf = p(s,5);
lamrf = p(s,6);
lamthetaf = p(s,7);
lamvrf = p(s,8);
lamvthetaf = p(s,9);
```

```
lammf = p(s,10);

% control
beta = atan2(lamvrf,lamvthetaf);

% hamiltonian
Hf = lamrf*vrf + lamthetaf*vthetaf/rf + lamvrf*(vthetaf^2/rf - param.mew/rf^2 + param.T*sin(beta)/mf)...
    + lamvthetaf*(param.T*cos(beta)/mf - vthetaf*vrf/rf) + lammf*(-param.T/param.ve);

% conditions
E = [ rf - param.rf;
      vrf - param.vrf;
      vthetaf - param.vthetaf;
      lamthetaf - param.lamthetaf;
      lammf - param.lammf;
      Hf - param.Hf];

end
```

Listing 3: orbitTransferOde.m

```
function pdot = orbitTransferOde(t,X,param,tf)

% given constants
mew = param.mew;
T = param.T;
ve = param.ve;

% states and co-states
r = X(1);
theta = X(2);
vr = X(3);
vtheta = X(4);
m = X(5);
lamr = X(6);
lamtheta = X(7);
lamvr = X(8);
lamvtheta = X(9);
lamm = X(10);

% control
beta = atan2(lamvr,lamvtheta);

% differential equations
rdot = vr;
thetadot = vtheta/r;
vrdot = vtheta^2/r - mew/r^2 + T*sin(beta)/m;
vthetadot = -vtheta*vr/r + T*cos(beta)/m;
mdot = -T/ve;
lamrdot = lamtheta*vtheta/r^2 + lamvr*(vtheta^2/r^2 - 2*mew/r^3) - lamvtheta*vr*vtheta/r^2;
lamthetadot = 0;
lamvrdot = -lamr + lamvtheta*vtheta/r;
lamvthetadot = -lamtheta/r - 2*lamvr*vtheta/r + lamvtheta*vr/r;
lammdot = lamvr*T*sin(beta)/m^2 + lamvtheta*T*cos(beta)/m^2;


pdot = [rdot;
        thetadot;
        vrdot;
        vthetadot;
        mdot;
        lamrdot;
        lamthetadot;
        lamvrdot;
        lamvthetadot;
        lammdot];
pdot = pdot * (tf - param.t0)/2;
```

```
end
```

Listing 4: midterm_indirect_mutli_shooting_main.m

```
%% EML6934 Optimal Control
% Name: Elias Reyes
% Date: 04 April 2022
% Assignment: Midterm
% Goal: Solve the orbit transfer problem using indirect multiple shooting
close all; clear all; clc
format longg
format compact

% specify number of intervals to loop through
k_list = [ 2 4 8 16];

% specify number of figures per case
num_figs = 2;

% intitialize performance vectors
numk = numel(k_list);
terminal_time = zeros(numk,1);
terminal_mass = zeros(numk,1);
iter = zeros(numk,1);
sim_time = zeros(numk,1);

% set num states
param.numStates = 10;

% given constants
param.T = 0.1405;
param.mew = 1;
param.ve = 1.8658344;

% boundary conditions
param.r0 = 1;
param.theta0 = 0;
param.vr0 = 0;
param.vtheta0 = sqrt(param.mew/param.r0);
param.m0 = 1;
param.t0 = 0;
param.rf = 1.5;
param.vrf = 0;
param.vthetaf = sqrt(param.mew/param.rf);

% transversality conditions
param.lamthetaf = 0;
param.lammf = 1;
param.Hf = 0;

% initial guesses for unkown co-states
lamr = -2;
lamtheta = 0;
lamvtheta = 2;
lamvr = 2;
lamm = 2;
tf = 3.5;

count = 1; % count for figs
% loop through all the cases
for kidx = 1:numel(k_list)
    % set specific interval case
    param.k = k_list(kidx);

    % Create conditions that need to be solved. This takes the intial guesses
    % above and repeats them for the desired number of intervals. This also sets
    % the guess for the states at each interval.
```

```
64      lamguess = [lamr; lamtheta; lamvtheta; lamvr; lamm];
65      stateguess = ones(param.numStates/2,1);
66      pguess = repmat([stateguess;lamguess],param.k-1,1); % state/cp-state guesses
67      tfguess = tf;
68      % tf guess
69      % concatinate all the unkowns. Begin with the only the costates since we know
70      % the states at t0
71      zguess = [lamguess; pguess; tfguess];
72
73      options = optimoptions('fsolve','Display','off','TolFun',1e-8);
74
75      % solve the optimal control problem
76      f = @(x)indirectMultiOrbitTransferError(x,param);
77
78      tic % start timer
79      [solution,~,~,output] = fsolve(f, zguess ,options);
80
81      elapsed_time = toc; % end timer and log
82      iterations = output.iterations;
83
84      %% Take optimal initial conditions and integrate them for plotting purposes
85      color = ['b','r','g','c','m'];
86      r0 = param.r0;
87      theta0 = param.theta0;
88      vr0 = param.vr0;
89      vtheta0 = param.vtheta0;
90      m0 = param.m0;
91
92      tf = solution(end); % optimal tf
93      % optimal conditions at each interval
94      zvec = [r0; theta0; vr0; vtheta0; m0; solution(1:end-1)];
95      % reshape to make number of columns equal to number of intervals. This
96      % helps with indexing.
97      zvec = reshape(zvec,param.numStates,param.k);
98      time = linspace(param.t0,tf,param.k+1);
99      options = odeset('reltol',1e-6);
100     tau = linspace(-1,1,param.k+1);
101     for idx = 1:param.k
102         X0 = zvec(:,idx);
103         tspan = [time(idx) time(idx+1)];
104         tspan = [tau(idx) tau(idx+1)];
105         [t,p] = ode113(@orbitTransferOde,tspan,X0,options,param,tf);
106
107         s = length(t);
108
109         lamvrf = p(:,8);
110         lamvthetaf = p(:,9);
111
112         % optimal control
113         beta = mod(atan2(lamvrf,lamvthetaf),2*pi);
114
115         % plot figures
116         figure(count); hold on;
117         for p_idx = 1:param.numStates/2
118             plot(t,p(:,p_idx),color(p_idx));
119             h = plot(t([1 end]),p([1 end],p_idx),'*k');
120             h.Annotation.LegendInformation.IconDisplayStyle = 'off';
121         end
122         xlabel('t','Interpreter','LaTeX')
123         legend('r(t)','$\theta(t)$','vr(t)','v$\theta(t)$','m(t)','Interpreter','LaTeX')
124         set(gcf,'color','white')
125         set(gca,'fontweight','bold','fontsize',10,'XMinorGrid','on','YMinorGrid','on')
126         str1 = sprintf('States for trajectory that minimized fuel consumption (%d Intervals)',param.k);
127         title(str1);
128
129         figure(count+1); hold on;
130         plot(t([1 end]),beta([1 end])*180/pi,'*k')
131         plot(t,beta*180/pi)
```

```
132             set(gcf,'color','white')
133             set(gca,'fontweight','bold','fontsize',10,'XMinorGrid','on','YMinorGrid','on')
134             xlabel('t','Interpreter','LaTeX')
135             ylabel('$\beta(t)$','Interpreter','LaTeX')
136             str1 = sprintf('Optimal Control that minimizes fuel consumption (%d Intervals)',param.k);
137             title(str1);
138         end
139         % save off the data
140         mf = p(end,5);
141         terminal_time(kidx) = tf;
142         terminal_mass(kidx) = mf;
143         iter(kidx) = iterations;
144         sim_time(kidx) = elapsed_time;
145
146         % incriment figure index
147         count = count + num_figs;
148     end
```

Listing 5: indirectMultiOrbitTransferError.m

```
1   function E = indirectMultiOrbitTransferError(z,param)
2   E = [];
3
4   % states at t0
5   r0 = param.r0;
6   theta0 = param.theta0;
7   vr0 = param.vr0;
8   vtheta0 = param.vtheta0;
9   m0 = param.m0;
10  % tf guess
11  tf = z(end);
12  % concatinate known and unknown states
13  zvec = [r0; theta0; vr0; vtheta0; m0; z(1:end-1)];
14  % reshape so that each column is an interval
15  zvec = reshape(zvec,param.numStates,param.k);
16
17  time = linspace(param.t0,tf,param.k+1);
18  options = odeset('reltol',1e-6);
19  tau = linspace(-1,1,param.k+1);
20  for idx = 1:param.k
21      % intial conditions at specifc interval
22      X0 = zvec(:,idx);
23      % tspan for specific interval
24      tspan = [time(idx) time(idx+1)];
25      tspan = [tau(idx) tau(idx+1)];
26      [t,p] = ode113(@orbitTransferOde,tspan,X0,options,param, tf);
27
28      s = length(t);
29
30      % states and co-states at end of interval
31      rf = p(s,1);
32      thetaf = p(s,2);
33      vrf = p(s,3);
34      vthetaf = p(s,4);
35      mf = p(s,5);
36      lamrf = p(s,6);
37      lamthetaf = p(s,7);
38      lamvrf = p(s,8);
39      lamvthetaf = p(s,9);
40      lammf = p(s,10);
41
42
43      if idx == param.k
44          % optimal control
45          beta = atan2(lamvrf,lamvthetaf);
46          % use optimal control and states/costates to get hamiltonian
47          Hf = lamrf*vrf + lamthetaf*vthetaf/rf + lamvrf*(vthetaf^2/rf - param.mew/rf^2 + param.T*sin(beta)/mf)...
```

```
            + lamvthetaf*(param.T*cos(beta)/mf - vthetaf*vrf/rf) + lammf*(-param.T/param.ve);
        % errros at final interval
        E_temp = [ rf - param.rf;
                   vrf - param.vrf;
                   vthetaf - param.vthetaf;
                   lamthetaf - param.lamthetaf;
                   lammf - param.lammf;
                   Hf - param.Hf ];
        % concatinate errors
        E = [E; E_temp];
    else
        % errors at each interval k-1
        pint = reshape(p(end,:),[],1);
        E_temp = pint - zvec(:,idx+1);
        % concatinate errors
        E = [E; E_temp];
    end
end
end
```

Listing 6: midterm_direct_shooting_main.m

```
%% EML6934 Optimal Control
% Name: Elias Reyes
% Date: 04 April 2022
% Assignment: Midterm
% Goal: Solve the orbit transfer problem using direct shooting
close all; clear all; clc
format longg
format compact

% specify number of coefficients to loop through
n_list = [ 2 3 4 5 6 ];

% specify number of figures per case
num_figs = 2;

% intitialize performance vectors
numn = numel(n_list);
terminal_time = zeros(numn,1);
terminal_mass = zeros(numn,1);
iter = zeros(numn,1);
sim_time = zeros(numn,1);

% set number of states in problem
param.numStates = 5; % r,theta,vr,vtheta,m
% given conditions
param.T = 0.1405;
param.mew = 1;
param.ve = 1.8658344;
% boundary conditions
param.r0 = 1;
param.theta0 = 0;
param.vr0 = 0;
param.vtheta0 = sqrt(param.mew/param.r0);
param.m0 = 1;
param.t0 = 0;
param.rf = 1.5;
param.vrf = 0;
param.vthetaf = sqrt(param.mew/param.rf);

count = 1; % counter for figures
% loop through different polynomial degrees cases
for idxn = 1:numn

    param.n = n_list(idxn); % degree of polynomial used to parameterize the control
    param.numCoeff = param.n+1; % always 1 more coefficent then degree
```

```
46
47      % initial guesses for unkown parameters
48      tfguess = 10;
49      cguess = zeros(param.numCoeff,1);
50      zguess = [cguess; tfguess];
51
52      % set up bounds
53      cmin = -50*ones(param.numCoeff,1);
54      cmax = 50*ones(param.numCoeff,1);
55      tfmin = 0;
56      tfmax = 100;
57      zmin = [cmin; tfmin];
58      zmax = [cmax; tfmax];
59      A = [];
60      B = [];
61      Aeq = [];
62      Beq = [];
63
64      options = optimset('MaxFunEvals',100000,'MaxIter',1000,'Display','off');
65
66      tic % start timer
67      [solution,~,~,output] = fmincon(@orbitTransferObj,zguess,A,B,Aeq,Beq,zmin,zmax,@directOrbitTransferError,options,param);
68
69      elapsed_time = toc; % end timer and log
70      iterations = output.iterations; % log number of iterations
71
72      %% Take optimal initial conditions and integrate them for plotting purposes
73      color = ['b','r','g','c','m'];
74      tf = solution(end); % optimal tf
75      c = solution(1:end-1); % optimalcoefficients
76
77      % initial conditions
78      X0 = [param.r0; param.theta0; param.vr0; param.vtheta0; param.m0];
79
80      % use tau scale [-1 1]
81      tau = [-1 1];
82      options = odeset('reltol',1e-6);
83
84      % propagate with the optimal conditions and coefficients to get the optimal trajectory
85      [t,p] = ode113(@directOrbitTransferOde,tau,X0,options,c,param,tf);
86
87      % otpimal control
88      beta = polyval(c,t)*180/pi;
89
90      % plot the optimal trajectories and control
91      figure(count); hold on; grid minor
92      for p_idx = 1:param.numStates
93          plot(t,p(:,p_idx),color(p_idx));
94          h = plot(t([1 end]),p([1 end],p_idx),'*k');
95          h.Annotation.LegendInformation.IconDisplayStyle = 'off';
96      end
97      xlabel('$\tau$','Interpreter','LaTeX')
98      legend('$r(\tau)$','$\theta(\tau)$','$v_r(\tau)$','$v_\theta(\tau)$','$m(\tau)$','Interpreter','LaTeX')
99      set(gcf,'color','white')
100     set(gca,'fontweight','bold','fontsize',10)
101     str1 = sprintf('States for trajectory that minimized fuel consumption');
102     title(str1);
103
104     figure(count +1); hold on; grid minor
105     plot(t([1 end]),beta([1 end]),'*k')
106     plot(t,beta)
107     set(gcf,'color','white')
108     set(gca,'fontweight','bold','fontsize',10)
109     xlabel('$\tau$','Interpreter','LaTeX')
110     ylabel('$\beta(\tau)$','Interpreter','LaTeX')
111     str1 = sprintf('Optimal Control that minimizes fuel consumption');
112     title(str1);
113
```

```
114     % save off the data
115     mf = p(end,5);
116     terminal_time(idxn) = tf;
117     terminal_mass(idxn) = mf;
118     iter(idxn) = iterations;
119     sim_time(idxn) = elapsed_time;
120
121     % incriment figure index
122     count = count + num_figs;
123 end
```

Listing 7: directOrbitTransferError.m

```
1  function [Eineq, E] = directOrbitTransferError(z,param)
2  Eineq = [];
3  E = [];
4
5  % guess of final time
6  tf = z(end);
7  % polynomial coefficients
8  c = z(1:end-1);
9  % initial conditions
10 X0 = [param.r0; param.theta0; param.vr0; param.vtheta0; param.m0];
11 % tau scale
12 tspan = [-1 1];
13 options = odeset('reltol',1e-6);
14 % solve ode with initial conditions and current guesses
15 [t,p] = ode113(@directOrbitTransferOde,tspan,X0,options,c,param,tf);
16
17 s = length(t);
18 % states at end of interval
19 rf = p(s,1);
20 vrf = p(s,3);
21 vthetaf = p(s,4);
22
23
24 E = [ rf - param.rf;
25       vrf - param.vrf;
26       vthetaf - param.vthetaf ];
27
28 end
```

Listing 8: directOrbitTransferOde.m

```
1  function pdot = directOrbitTransferOde(t,X,c,param,tf)
2
3  % control
4  beta = polyval(c,t);
5
6  % given constants
7  mew = param.mew;
8  T = param.T;
9  ve = param.ve;
10
11 % states
12 r = X(1);
13 theta = X(2);
14 vr = X(3);
15 vtheta = X(4);
16 m = X(5);
17
18 % differential equations
19 rdot = vr;
20 thetadot = vtheta/r;
21 vrdot = vtheta^2/r - mew/r^2 + T*sin(beta)/m;
22 vthetadot = -vtheta*vr/r + T*cos(beta)/m;
```

```
mdot = -T/ve;

pdot = [rdot;
        thetadot;
        vrdot;
        vthetadot;
        mdot];

% need to multiply by (tf-t0)/2 since I am on the tau [-1 1] scale
pdot = pdot * (tf - param.t0)/2;
end
```

Listing 9: orbitTransferObj.m

```
function J = orbitTransferObj(z,~)
%minimize time
tf = z(end);
J = tf;
```

Listing 10: midterm_direct_mutli_shooting_main.m

```
%% EML6934 Optimal Control
% Name: Elias Reyes
% Date: 04 April 2022
% Assignment: Midterm
% Goal: Solve the orbit transfer problem using direct multiple shooting
close all; clear all; clc
format longg
format compact

% specify desired polynomial degrees and intervals
n_list = [2 3 4 5 6];
k_list = [2 4 8 16];

% specify number of figures in loop
num_figs = 2;

% initialize performance matrices
numk = numel(k_list);
numn = numel(n_list);
terminal_time = zeros(numn,numk);
terminal_mass = zeros(numn,numk);
iter = zeros(numn,numk);
sim_time = zeros(numn,numk);

% set number of states
param.numStates = 5; % r,theta,vr,vtheta,m

%known conditions
param.T = 0.1405;
param.mew = 1;
param.ve = 1.8658344;

% boundary conditions
param.r0 = 1;
param.theta0 = 0;
param.vr0 = 0;
param.vtheta0 = sqrt(param.mew/param.r0);
param.m0 = 1;
param.t0 = 0;
param.rf = 1.5;
param.vrf = 0;
param.vthetaf = sqrt(param.mew/param.rf);

% intitialize counter for plots
count = 1;
```

```
46
47  for idxn = 1:numn
48
49      % degree of polynomial used to parameterize the control
50      param.n = n_list(idxn);
51      param.numCoeff = param.n+1; % always 1 more coefficent then degree
52
53      for idxk = 1:numk
54
55          % number of intervals
56          param.k = k_list(idxk);
57
58          % initial guesses for unkown parameters
59          tfguess = 10;
60          pguess = ones(param.numStates*(param.k-1),1);
61          cguess = zeros(param.numCoeff*param.k,1);
62          zguess = [pguess; cguess; tfguess];
63
64          % set up bounds
65          pmin = -5*ones(numel(pguess),1);
66          pmax = 5*ones(numel(pguess),1);
67          cmin = -50*ones(param.numCoeff*param.k,1);
68          cmax = 50*ones(param.numCoeff*param.k,1);
69          tfmin = 0;
70          tfmax = 11;
71          zmin = [pmin; cmin; tfmin];
72          zmax = [pmax; cmax; tfmax];
73          A = [];
74          B = [];
75          Aeq = [];
76          Beq = [];
77
78          options = optimset('MaxFunEvals',100000,'MaxIter',1000,'Display','off','TolFun',1e-4);
79
80          tic % starter timer
81          [solution,~,~,output] = fmincon(@orbitTransferObj,zguess,A,B,Aeq,Beq,zmin,zmax,@directMultiOrbitTransferError,options,param);
82
83          elapsed_time = toc; % stop timer and record
84          iterations = output.iterations; % record num iterations
85
86          %% Take optimal conditions and integrate them for plotting purposes
87          color = ['b','r','g','c','m'];
88          % states at t0
89          r0 = param.r0;
90          theta0 = param.theta0;
91          vr0 = param.vr0;
92          vtheta0 = param.vtheta0;
93          m0 = param.m0;
94          P0 = [r0; theta0; vr0; vtheta0; m0];
95
96          % seperate the unknowns
97          % P is the unknown states
98          P_end = param.numStates*(param.k-1);
99          P_tmp = solution(1:P_end);
100         P = [P0;P_tmp];
101         P = reshape(P,param.numStates,[]); % reshape to make each column an interval
102         % c is the uknown coeffients
103         c_end = numel(solution)-1;
104         c_list = solution(P_end+1:c_end);
105         c_list = reshape(c_list,param.numCoeff,[]); % reshape to make each column an interval
106         tf = solution(end);
107
108         % create tau grid
109         tau = linspace(-1,1,param.k+1);
110         options = odeset('reltol',1e-6);
111
112         for idx = 1:param.k
113             % extract coefficents for the specific interval
```

```
114                 c = c_list(:,idx);
115                 X0 = P(:,idx);
116                 % tau span for specific interval
117                 tspan = [tau(idx) tau(idx+1)];
118                 [t,p] = ode113(@directOrbitTransferOde,tspan,X0,options,c,param,tf);
119
120                 % optimal control
121                 beta = mod(polyval(c,t),2*pi);
122
123                 % plot figures
124                 figure(count); hold on;
125                 for p_idx = 1:param.numStates
126                     plot(t,p(:,p_idx),color(p_idx));
127                     h = plot(t([1 end]),p([1 end],p_idx),'*k');
128                     h.Annotation.LegendInformation.IconDisplayStyle = 'off';
129                 end
130                 xlabel('$\tau$','Interpreter','LaTeX')
131                 legend('$r(\tau)$','$\theta(\tau)$','$v_r(\tau)$','$v_\theta(\tau)$','$m(\tau)$','Interpreter','LaTeX')
132                 set(gcf,'color','white')
133                 set(gca,'fontweight','bold','fontsize',10,'XMinorGrid','on','YMinorGrid','on')
134                 str1 = sprintf('States for trajectory that minimized fuel consumption (%d Intervals)',param.k);
135                 title(str1);
136
137                 figure(count+1); hold on;
138                 plot(t([1 end]),beta([1 end])*180/pi,'*k')
139                 plot(t,beta*180/pi)
140                 set(gcf,'color','white')
141                 set(gca,'fontweight','bold','fontsize',10,'XMinorGrid','on','YMinorGrid','on')
142                 xlabel('$\tau$','Interpreter','LaTeX')
143                 ylabel('$\beta(\tau)$','Interpreter','LaTeX')
144                 str1 = sprintf('Optimal Control that minimizes fuel consumption (%d Intervals)',param.k);
145                 title(str1);
146             end
147
148             % save off data
149             mf = p(end,5);
150             terminal_time(idxn,idxk) = tf;
151             terminal_mass(idxn,idxk) = mf;
152             iter(idxn,idxk) = iterations;
153             sim_time(idxn,idxk) = elapsed_time;
154
155             % incriment figure index
156             count = count + num_figs;
157         end
158     end
159
160     %% Put all the data into a stuct
161     fields = {'N','K','Iterations','SimTime','TerminalTime','TerminalMass'};
162     count = 1;
163     for idxn = 1:numn
164         N = n_list(idxn);
165         for idxk = 1:numk
166             fieldname = sprintf('combo%d',count);
167             K = k_list(idxk);
168             itr = iter(idxn,idxk);
169             t = sim_time(idxn,idxk);
170             tf = terminal_time(idxn,idxk);
171             mf = terminal_mass(idxn,idxk);
172             stuff = [N K itr t tf mf];
173             for kfield = 1:numel(fields)
174                 junk = stuff(kfield);
175                 data.(fieldname).(fields{kfield}) = junk;
176             end
177             count = count + 1;
178         end
179     end
180
181     %% Additional Figures
```

```
nlistvec = repmat(n_list',1,numel(k_list));
klistvec = repmat(k_list,numel(n_list),1);

figure;
surf(nlistvec,klistvec,sim_time, 'FaceAlpha',0.75)
xlabel('Polynomial Degree N')
ylabel('Numer of Intervals K')
zlabel('Execution Time')
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10, 'XMinorGrid','on','YMinorGrid','on')
title('Direct Multiple Shooting Performance')

figure;
surf(nlistvec,klistvec,terminal_time, 'FaceAlpha',0.75)
xlabel('Polynomial Degree N')
ylabel('Numer of Intervals K')
zlabel('Terminal Time')
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10, 'XMinorGrid','on','YMinorGrid','on')
title('Direct Multiple Shooting Objective Function Performance')

figure; hold on;
for idx = 1:numn
plot(klistvec(idx,:),sim_time(idx,:),'-o')
end
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10, 'XMinorGrid','on','YMinorGrid','on')
xlabel('Intervals')
ylabel('Execution Time')
legend('Degree 2','Degree 3','Degree 4','Degree 5','Degree 6')
title('Execution Performance for Direct Multi Shooting')


figure; hold on;
str = sprintf('IS');
plot([0 1],[1 t_mass_IS],'DisplayName',str)
for idx = 1:numel(t_mass_IMS)
    str = sprintf('IMS K-%d',k_list(idx));
    plot([0 1],[1 t_mass_IMS(idx)],'DisplayName',str)
end
for idx = 1:numel(t_mass_DS)
    str = sprintf('DS N-%d',n_list(idx));
    plot([0 1],[1 t_mass_DS(idx)],'DisplayName',str)
end
for idx = 1:numn
    for idx1 = 1:numk
        str = sprintf('DMS N-%d K-%d',n_list(idx),k_list(idx1));
        plot([0 1],[1 terminal_mass(idx,idx1)],'DisplayName',str)
    end
end
legend
set(gcf,'color','white')
set(gca,'fontweight','bold','fontsize',10, 'XMinorGrid','on','YMinorGrid','on')
title('Optimized Terminal Mass For IS, IMS, DS, and DMS')
ylabel('$m$','Interpreter','LaTeX')
xlabel('$\tau$','Interpreter','LaTeX')


terminal_mass =...
        [0.755354731430044, 0.755431874367474, 0.755435693528483, 0.755442644030903;
        0.755417615837166, 0.755431353949163, 0.755432919596403, 0.755441577328962;
        0.755398775622557, 0.755427047544431, 0.755426446694972, 0.755458867973107;
        0.755381642127587, 0.755424606318781, 0.755433590252236, 0.755452011992764;
        0.755373228394507, 0.755423618571835, 0.75543765370842, 0.755450844950145];
t_mass_IMS = [0.755500533918634;
        0.755500513060439;
        0.755500474729086;
        0.755500456162572];
```

```
t_mass_DS = [0.708869823160571;
             0.735624993263296;
             0.738533068696249;
             0.746677283977104;
             0.746901711579395];
t_mass_IS = 0.755500499843343;
```

Listing 11: directMultiOrbitTransferError.m

```
function [Eineq, E] = directMultiOrbitTransferError(z,param)
E = [];
Eineq = [];

% initial conditions
r0 = param.r0;
theta0 = param.theta0;
vr0 = param.vr0;
vtheta0 = param.vtheta0;
m0 = param.m0;
P0 = [r0; theta0; vr0; vtheta0; m0];

% seperate the unknowns
% P is the unknown states
P_end = param.numStates*(param.k-1);
P_tmp = z(1:P_end);
P = [P0;P_tmp];
P = reshape(P,param.numStates,[]); % reshape to make each column an interval
% c is the unknown coeffients for polynomial
c_end = numel(z)-1;
c_list = z(P_end+1:c_end);
c_list = reshape(c_list,param.numCoeff,[]); % reshape to make each column an interval
tf = z(end);

% create grid to integrate on tau [-1 1]
tau = linspace(-1,1,param.k+1);
options = odeset('reltol',1e-6);

for idx = 1:param.k
    % extract coefficents for the specific interval
    c = c_list(:,idx);
    % extract states for the specific interval
    X0 = P(:,idx);
    % specific integartion span
    tspan = [tau(idx) tau(idx+1)];
    % solve for conditions at the end of the interval
    [t,p] = ode113(@directOrbitTransferOde,tspan,X0,options,c,param,tf);

    s = length(t);

    % states at end of interval
    rf = p(s,1);
    vrf = p(s,3);
    vthetaf = p(s,4);

    % get the errors
    if idx == param.k
        % errors at final interval
        E_temp = [ rf - param.rf;
                   vrf - param.vrf;
                   vthetaf - param.vthetaf ];

        E = [E; E_temp];
    else
        % errors at each interval k-1
        pint = reshape(p(end,:),[],1);
        E_temp = pint - P(:,idx+1);

```

```
59              E = [E; E_temp];
60          end
61      end
62      end
```