Demonstration of GPOPS-II for Optimal Configuration of a Tetrahedral Spacecraft Formation

Author: Mallory Daly

Supervisor: Prof. Anil Rao

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science with Honors in the Flight Dynamics and Optimization Laboratory Department of Mechanical and Aerospace Engineering

April 2015
1 Abstract

The general purpose optimization solver, GPOPS-II, is used to solve an optimal control problem in which mass is minimized. Four small spacecraft begin at the same initial position in a circular orbit and use one maneuver to reach a terminal reference orbit, burning a maximum of 300 kg of fuel; the spacecraft form a tetrahedron of certain volumetric constraints at the apogee the terminal reference orbit. The methods used to formulate the problem in MATLAB® are discussed. The problem is solved at several benchmarks, beginning with just one spacecraft and ending with the complete problem. It was found that 163.25 kg of fuel was needed for one spacecraft to ascend to the terminal reference orbit and 653.95 kg of fuel for four spacecraft to ascend to the terminal reference orbit and form the tetrahedron.
2 CONTENTS

1 Abstract ................................................................................................................................. 2
2 Contents ............................................................................................................................... 3
3 Physical Constants ............................................................................................................... 5
4 Symbols ............................................................................................................................... 6
5 Introduction .......................................................................................................................... 7
6 Detailed Description of the Problem ....................................................................................... 8
  6.1 Spacecraft Parameters ...................................................................................................... 9
  6.2 Initial Conditions ............................................................................................................. 10
  6.3 Final Conditions ............................................................................................................ 10
  6.4 Dynamics ....................................................................................................................... 11
    6.4.1 Coast Phase ............................................................................................................. 11
    6.4.2 Burn Phase ............................................................................................................ 11
  6.5 Path Constraints ........................................................................................................... 12
  6.6 Event Constraints ......................................................................................................... 12
    6.6.1 Phase Linkages ....................................................................................................... 12
    6.6.2 Tetrahedron Shape Constraints .......................................................................... 13
    6.6.3 Repeatability Constraint ....................................................................................... 13
  6.7 Assumptions .................................................................................................................. 13
7 Methods .............................................................................................................................. 14
  7.1 Scaling ........................................................................................................................... 14
  7.2 Boundary Conditions .................................................................................................... 14
  7.3 Stepwise Coding and Initial Guesses .......................................................................... 15
  7.4 Selecting the NLP Solver ............................................................................................. 15
  7.5 Coding the Problem in MATLAB® ............................................................................... 16
    7.5.1 Considering the Phase Definitions for Four Spacecraft ....................................... 16
    7.5.2 Finding the Mesocenter ....................................................................................... 16
    7.5.3 Calculating the Side Lengths .............................................................................. 17
    7.5.4 Calculating the Ideal and Actual Volumes ....................................................... 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5.5</td>
<td>Finding the Final Semimajor Axis of Each Spacecraft</td>
<td>18</td>
</tr>
<tr>
<td>7.5.6</td>
<td>Setting the Objective</td>
<td>18</td>
</tr>
<tr>
<td>7.6</td>
<td>Plotting Functions</td>
<td>18</td>
</tr>
<tr>
<td>7.7</td>
<td>Troubleshooting Techniques</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Results</td>
<td>20</td>
</tr>
<tr>
<td>8.1</td>
<td>One Spacecraft</td>
<td>20</td>
</tr>
<tr>
<td>8.2</td>
<td>Four Spacecraft</td>
<td>22</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Mesocenter and Equal Final Times Constrained</td>
<td>22</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Mesocenter, Equal Final Times, and Side Length Constrained</td>
<td>22</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Mesocenter, Equal Final Times, Side Length, and Geometric Factor Constrained</td>
<td>22</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Final Solution (All Constraints Included)</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>References</td>
<td>27</td>
</tr>
<tr>
<td>Appendix A</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>Appendix B</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Appendix C</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>Appendix D</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Appendix E</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Appendix F</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>Appendix G</td>
<td></td>
<td>55</td>
</tr>
</tbody>
</table>
3 PHYSICAL CONSTANTS

Acceleration due to gravity \( g_0 \) = 9.80665 m/s\(^2\)
Standard gravitational parameter of Earth \( \mu \) = 3.986004415 \times 10^{14} \text{ m}^3/\text{s}^2
Radius of Earth \( R_E \) = 6378145 m
4 SYMBOLS

\[ h \]  Altitude  \[ m \]
\[ R_a \]  Distance to apogee measured from the center of the Earth  \[ m \]
\[ R_p \]  Distance to perigee measured from the center of the Earth  \[ m \]
\[ m \]  Mass  \[ kg \]
\[ T \]  Thrust  \[ N \]
\[ t \]  Time  \[ s \]
\[ I_{SP} \]  Specific impulse  \[ s \]

**Directional Control**
\[ u_r \]  Control magnitude in the radial direction
\[ u_\theta \]  Control magnitude in the tangential direction
\[ u_n \]  Control magnitude in the normal direction
\[ a_r \]  Perturbing acceleration in the radial direction  \[ N/m \]
\[ a_\theta \]  Perturbing acceleration in the tangential direction  \[ N/m \]
\[ a_n \]  Perturbing acceleration in the normal direction  \[ N/m \]

**Cartesian Coordinates**
\[ r \]  Position vector measured from the center of the Earth  \[ m \]
\[ v \]  Velocity vector measured from the center of the Earth  \[ m/s \]

**Classical Orbital Elements**
\[ a \]  Semimajor axis  \[ m \]
\[ e \]  Eccentricity
\[ i \]  Inclination  \[ deg \]
\[ \omega \]  Argument of perigee  \[ deg \]
\[ \Omega \]  Longitude of the ascending node  \[ deg \]
\[ v \]  True anomaly  \[ deg \]

**Modified Equinoctial Elements**
\[ p \]  Semilatus rectum  \[ m \]
\[ P_1, P_2, Q_1, Q_2 \]  States of no geometric definition
\[ L \]  True longitude  \[ deg \]
5 Introduction

The purpose of this research is to demonstrate the use of GPOPS-II [1], a general purpose optimization software, to solve a scientifically-relevant optimal control problem, the optimal configuration of a tetrahedral spacecraft formation. The problem is adopted from Geoffrey Huntington, David Benson, and Anil Rao’s Optimal Configuration of Tetrahedral Spacecraft Formations [2] in order to validate the results obtained with the GPOPS-II software. The writing of the methods in this thesis is designed to be applicable to any user of GPOPS-II solving a similar problem.

In the problem, four small spacecraft begin from the same position of a circular orbit in a coast phase, imitating the release of the spacecraft from their carrier. The spacecraft maneuver to a reference orbit using one burn phase, during which the thrust of each craft is controlled in the radial, tangential, and normal directions. The crafts then coast to the apogee of the reference orbit so that their final configuration forms a tetrahedron of certain volumetric constraints. Forming the tetrahedron while in the coast phase, rather than a burn phase, allows for the use of scientific instruments. The controls are optimized in order to maximize final mass of the spacecraft (minimize fuel use).

This problem was selected for its relevance to small satellite missions. The four NASA Magnetospheric Multiscale (MMS) spacecraft are currently employing an orbital configuration in the form of a pyramid for the study of magnetic reconnection [3]. The scale of these spacecraft is the same as the crafts used in the tetrahedral problem: the mass of propellant available for each MMS craft is 360 kg, in comparison to 300 kg used in the tetrahedral formation problem. Further, there is growing interest in using even smaller crafts, CubeSats, in formation flying applications. This is witnessed by many recent CubeSat missions utilizing or demonstrating formation flying, such as the QB50 [4], LoneStar-2 [5], and TanDEM-X [6]. Thus it was anticipated that this problem, or a variation of it, would be of interest for the flight planning and design of future formation missions.

GPOPS-II operates using variable-order adaptive orthogonal collocation methods and sparse nonlinear programming in order to solve broad optimal control problems [7]. Its platform is MATLAB®, and it requires the user to write three MATLAB® files: (a) an endpoint function, (b) a continuous function, and (c) a main script. The endpoint function defines the event constraints and the cost function to be minimized. The continuous function defines the states, controls, path, parameters, and dynamics of each phase. The main script uses defined structures in order to set the lower and upper limits on the following quantities: (a) the time at the start and end of a phase; (b) the state at the start of a phase, during a phase, and at the end of a phase; (c) the control during a phase; (d) the path constraints; (e) the event constraints; and (f) the static parameters [7]. In addition, the main script is used to define constants that are passed to the functions, define the setup for the nonlinear programmer (NLP), and call GPOPS-II to solve the problem.
6 DETAILED DESCRIPTION OF THE PROBLEM

As mentioned in the Introduction, the problem involves three phases for each spacecraft: coast, burn, and coast. A conceptual illustration of the flight paths of the crafts is shown in Fig. 1. Note that using three phases is a simplification of [2], which allowed an extra burn and coast phase. This is a reasonable simplification, for the authors of [2] discovered that comparable results could be obtained using one maneuver (three phases) or two maneuvers (five phases).

Fig. 1. The four spacecraft begin from the same initial position. The crafts each have three phases (coast, burn, and coast) to reach the apogee of the final orbit and form the tetrahedron. Neither the time nor the location of the shift between phases is constrained, so the marks between the phases are for illustration only. The figure is not to scale.
The spacecraft are assumed to start from the same initial position. In the terminal condition, the mesocenter of the tetrahedral formation is at the apogee of the terminal reference orbit, where the mesocenter is defined as the position central to the locations of the four spacecraft. The terminal reference orbit is known by its apogee and perigee, where the apogee is the desired final altitude plus the Earth’s radius and the perigee is the initial altitude plus the Earth’s radius.

The radial, tangential, and normal directions, which are used to control the thrust direction, are shown schematically in Fig. 2.

![Diagram](image)

**Fig. 2.** The radial ($u_r$), tangential ($u_t$), and normal ($u_n$) directions are shown schematically. The normal direction is out of the page.

There are three reference systems used in this problem. Classical orbital elements [8] are used to define the initial and terminal conditions. Modified equinoctial elements are used to describe the system dynamics in order to remove the singularity in the orbital elements at $e = 0$ [9]. Finally, the Earth-centered, Earth-fixed (ECEF) Cartesian coordinate system [10] is used to place constraints on the tetrahedron’s shape and define the final position and velocity of the mesocenter. Defining the final position and velocity vectors constrain all terminal conditions from the classical orbital elements.

### 6.1 Spacecraft Parameters

The spacecraft parameters selected are typical of a standard apogee kick motor [11]. Each spacecraft has a dry mass of $m_{dry} = 200$ kg and a fuel mass of $m_{fuel} = 300$ kg. Therefore the initial mass is defined as
\[ m_0 = m_{\text{dry}} + m_{\text{fuel}} = 500 \text{ kg} \]  \hspace{1cm} (1)

The engine has a maximum thrust of \( T = 7015 \text{ N} \) and a specific impulse of \( I_{SP} = 285.7 \text{ s} \).

### 6.2 Initial Conditions

The initial conditions are taken from [2] and summarized here for convenience. The initial altitude is \( h_0 = 600 \text{ km} \). The classical orbital elements that define the initial condition are as follows:

\[
\begin{align*}
a_0 &= R_E + h_0 = 6978145 \text{ m} \\
e_0 &= 0 \\
i_0 &= 28^\circ \\
o_0 &= 270^\circ \\
\Omega_0 &= 0 \\
v_0 &= 270^\circ
\end{align*}
\hspace{1cm} (2)

The elements in (2) are converted to modified equinoctial elements using the transformation given in Appendix A. The MATLAB® code for the conversions is shown in Appendix B.

### 6.3 Final Conditions

The terminal conditions are similarly taken from [2] and summarized here for convenience. The final altitude is \( h_f = 7000 \text{ km} \). Thus the distance to the apogee is \( R_a = R_E + h_f = 13378145 \text{ m} \). The final semimajor axis and eccentricity are found using orbital mechanics [12], as described below. The terminal inclination, argument of perigee, and longitude of ascending node are chosen to be the same as their initial values. The final true anomaly is known as \( 180^\circ \) since it is measured from the direction of the perigee and we are defining the final position of the mesocenter at the apogee.

\[
\begin{align*}
a_f &= \frac{R_a + R_p}{2} = \frac{R_a + a_0}{2} = 10178145 \text{ m} \\
e_f &= \frac{R_a - R_p}{R_a + R_p} = \frac{R_a - a_0}{R_a + a_0} = 0.3144 \\
i_f &= i_0 = 28^\circ \\
o_f &= o_0 = 270^\circ \\
\Omega_f &= \Omega_f = 0 \\
v_f &= 180^\circ
\end{align*}
\hspace{1cm} (3)

As before, the elements in (3) are converted to modified equinoctial elements using the transformation given in Appendix A.
6.4 Dynamics

All of the dynamics are in modified equinoctial elements and taken from [13]. It is noted that in [2], the final coast phase is described using Cartesian coordinates. However, because only the last point of this phase is constrained, it was decided to keep the final coast phase dynamics in modified equinoctial elements. This made coding the problem more straightforward.

6.4.1 Coast Phase

Only the true longitude is changing during the coast phase, as described by (4), where \( p, P_1 \), and \( P_2 \) are constant values. It is noted that in the initial coast phase the values of \( p, P_1 \), and \( P_2 \) are known from the initial conditions. In the final coast phase, however, these values are not known and are input as parameters. An event constraint that links the parameters to the final values of \( p, P_1 \), and \( P_2 \) in the burn phase is used to solve for the parameters’ values.

\[
\dot{L} = \frac{dL}{dt} = \frac{\sqrt{\mu p}}{p^2} (1 + P_1 \sin L + P_2 \cos L)^2
\] (4)

6.4.2 Burn Phase

The burn phase dynamics use the following shorthand:

\[
w = 1 + P_1 \sin L + P_2 \cos L
\]
\[
s^2 = 1 + Q_1^2 + Q_2^2
\] (5)

Further, the perturbing accelerations due to the control in the radial, tangential, and normal directions are given in (6).

\[
a_r = \frac{T}{m} u_r
\]
\[
a_\theta = \frac{T}{m} u_\theta
\]
\[
a_n = \frac{T}{m} u_n
\] (6)

Thus the burn phase dynamics are described as in (7)

\[
p = \frac{2p}{w} \sqrt{p \frac{\mu}{\mu} a_\theta}
\]
\[
\dot{P}_1 = \sqrt{p \frac{\mu}{\mu} \left(-a_r \cos L + \frac{a_\theta}{w} ((w + 1) \sin L + P_1) + \frac{P_2 a_n}{w} (Q_2 \sin L - Q_1 \cos L)\right)}
\] (7)
\[ \dot{Q}_1 = \sqrt{\frac{p}{\mu}} \left( \frac{s^2}{2w} \right) a_n \sin L \]
\[ \dot{L} = \sqrt{\mu} \frac{(w)}{p} + \sqrt{\frac{P}{\mu}} (Q_2 \sin L - Q_1 \cos L) \]
\[ \dot{m} = -\frac{T}{g_0 I_{sp}} \]

### 6.5 Path Constraints

Two path constraints are explicitly defined. The first path constraint is that the vector of the thrust controls \( \mathbf{u} = [u_r, u_\theta, u_n] \) must be of unit length.

\[ \mathbf{u} \cdot \mathbf{u} = 1 \]

The next path constraint is that the mass of the spacecraft at any point cannot be below the dry mass of the spacecraft.

### 6.6 Event Constraints

The event constraints are used for three purposes: (a) to link states, parameters, and time together; (b) to add shape constraints to the tetrahedron; and (c) to add a repeatability constraint.

#### 6.6.1 Phase Linkages

The first and second phases of each spacecraft are linked by the constraints in (9), where the superscript in the parentheses represents the phase and the subscript 0 or f represent the initial or final point, respectively, in the phase.

\[ L_0^{(2)} - L_f^{(1)} = 0 \]
\[ t_0^{(2)} - t_f^{(1)} = 0 \]

Similarly, the second and third phases of each spacecraft are linked by the constraints in (10), where it is noted that \( p, P_1, \) and \( P_2 \) are constant parameters in the third phase.

\[ p^{(3)} - p_f^{(2)} = 0 \]
\[ p_1^{(3)} - p_1f^{(2)} = 0 \]
\[ p_2^{(3)} - p_2f^{(2)} = 0 \]
\[ L_0^{(3)} - L_f^{(2)} = 0 \]
\[ t_0^{(3)} - t_f^{(2)} = 0 \]

\[ (10) \]
Finally, in order for the tetrahedron to be formed, the final time of the last phase of each spacecraft must be the same. This is represented by (11), where the superscript in the brackets represents the spacecraft.

\[ t_f^{(3)[1]} = t_f^{(3)[2]} = t_f^{(3)[3]} = t_f^{(3)[4]} \]  

(11)

### 6.6.2 Tetrahedron Shape Constraints

There are two shape constraints on the tetrahedron, as restated from [2] for convenience. The first constraint is that the each side length, \( L_{ij} \), is between 4 km and 18 km. This distance is an acceptable range for scientific return on the NASA MMS mission [3]. This constraint is a stricter interpretation of the acceptable range than used in [2], which constrained the average side length.

\[ 4 \text{ km} \leq L_{ij} \leq 18 \text{ km} \]  

(12)

The second constraint is based on a volumetric metric called the \( Q_{RB} \) Geometric Factor [14]. This factor is defined as (13), where \( V_a \) is the actual volume of the tetrahedron and \( V^* \) is the ideal volume, or the volume of a regular tetrahedron with a side length the length of the average side length, \( L_{avg} \).

\[ Q_{RB} = \frac{V_a}{V^*} \]  

(13)

The constraint imposed is given by (14).

\[ Q_{RB} \geq 0.9 \]  

(14)

### 6.6.3 Repeatability Constraint

The period of a spacecraft is determined by its semimajor axis [15]. Thus, in order to allow for repeatability, the final semimajor axis of each spacecraft must be equivalent.

\[ a_f^{(3)[1]} = a_f^{(3)[2]} = a_f^{(3)[3]} = a_f^{(3)[4]} \]  

(15)

### 6.7 Assumptions

The following assumptions are made in the formulation of this problem:

1. Spherical Earth gravity,
2. The spacecraft may start from the same physical location,
3. The thrust is of constant magnitude,
4. The thrusting maneuvers are non-impulsive, and
5. There is no impedance from orbital debris.
7 METHODS

This section describes the process used to generate the MATLAB® code and solve the problem using GPOPS-II. Because setting up the optimization problem can be challenging, these methods are written to be useful to other users.

7.1 SCALING

The optimization performance can be influenced significantly by the problem’s scaling, especially if there is substantial unbalance in the problem [16]. For example, in this problem, the numerical values of the standard gravitational parameter of Earth ($3.986 \times 10^{14}$) and initial mass (500) are twelve orders of magnitude apart. GPOPS-II has built-in options for automatic scaling [7]. However, the problem had difficulty converging when it was originally coded without manual scaling. Adding manual scaling to the problem greatly improved the speed and accuracy with which GPOPS-II solved the problem.

Three scaling parameters can be used to scale all values: length, mass and time. The scaling parameters for length ($\bar{L}$), mass ($\bar{M}$), and time ($\bar{T}$) were determined as given in (16).

$$\bar{L} = R_E = 6378145 \text{ m}$$
$$\bar{M} = m_0 = 500 \text{ kg}$$
$$\bar{T} = \sqrt{\frac{R_E^3}{\mu}} = 801.8126 \text{ s}$$ (16)

All variables in the problem were then scaled using the appropriate scaling parameter or combination of scaling parameters. An example is shown in (17).

$$\mu_{scaled} = \frac{\tau^2}{L^3} = 1.0$$ (17)

7.2 BOUNDARY CONDITIONS

While the initial and final conditions are given in the problem, only the path constraints on the controls and mass (see Path Constraints) were provide as boundary conditions on the intermediary states in [2]. Thus consideration was given to selecting boundary conditions that were appropriate but generous, so that the problem was not unnecessarily constrained.

Selecting appropriate boundary conditions was complicated by the modified equinoctial element form of the dynamics, since the states $P_1, P_2, Q_1,$ and $Q_2$ have no geometric definition. Thus in order to determine the boundary conditions, the transformation equations from orbital elements to modified equinoctial elements, given in Appendix A, were considered. For each modified equinoctial element state, generous
minimum and maximums were selected based on the minimum and maximum values of the orbital elements that form the transformation. The selected values are given in (18).

\[
\begin{align*}
    p_{\text{min}} &= 0.5a_0 \\
    p_{\text{max}} &= 1.5a_f \\
    P_{1\text{min}} &= -1 \\
    P_{2\text{max}} &= 1 \\
    P_{2\text{min}} &= -1 \\
    P_{2\text{max}} &= 1 \\
    Q_{1\text{min}} &= -1 \\
    Q_{2\text{max}} &= 1 \\
    L_{\text{min}} &= -10\pi \\
    L_{\text{max}} &= 10\pi
\end{align*}
\]

(18)

### 7.3 Stepwise Coding and Initial Guesses

The next tactic employed was stepwise coding of the problem. That is, the problem was solved at benchmarked steps to verify that the code was converging and the output gave reasonable results. The results obtained at each benchmark are shown in the Results section. Furthermore, using stepwise coding allowed for the generation of successively better initial guesses. The optimization method finds the local, not global, solution. Therefore, the initial guesses have a significant impact on whether the best solution is found, or whether or not the program converges to an optimal solution at all.

The solution at each step was saved in a MAT-file by saving the workspace. A function was written in MATLAB® in order to easily extract the guesses of a previously saved solution to provide the guesses in the main file of the next iteration. The function (extractGuesses.m) is shown in Appendix D.

### 7.4 Selecting the NLP Solver

There are two NLP solvers available to use with GPOPS-II. The first NLP solver is IPOPT (“Interior Point Optimizer”). IPOPT operates using an interior-point line-search filter method, which makes it particularly suitable for large problems with many variables and constraints [17]. The MATLAB files needed for IPOPT are included with the GPOPS-II package and the default error tolerance for IPOPT is \(10^{-7}\) [7].

The second NLP solver is SNOPT (“Sparse Nonlinear Optimizer”), which employs a sparse sequential quadratic programming algorithm and limited-memory quasi-Newton approximations. SNOPT is most effective for problems with large numbers of sparse linear constraints and a nonlinear objective function [18]. The SNOPT package is not included with GPOPS-II, but may be downloaded from [18]. Its default error tolerance is \(10^{-6}\) [7], making it slightly less accurate than IPOPT.
Alternating the NLP solver can be a method to troubleshoot a non-converging problem. IPOPT was utilized until the final form of the problem (four spacecraft with all event constraints), at which point the problem would not converge. When the NLP solver was changed to SNOPT, an optimal solution was found.

7.5 CODING THE PROBLEM IN MATLAB®

The format of the main file, endpoint function, and continuous function, briefly described in the Introduction, is given in detail in [7]. The problem was first coded for a singular spacecraft. Since there is only one spacecraft, the tetrahedron shape constraints, repeatability constraint, and final time equality constraint are irrelevant at this benchmark. The code for all three files of the one spacecraft problem is given in Appendix E.

Next the four spacecraft problem was coded. The constraints in the function were added iteratively following the stepwise coding tactic described previously. The final code is shown in Appendix F. Selected conceptualizations used for coding for the four spacecraft problem are described in detail in the sections below.

7.5.1 Considering the Phase Definitions for Four Spacecraft

The one spacecraft problem has three phases (coast, burn, and coast). To expand the problem to four spacecraft, one technique is to quadruple the state variables in each phase. However, adding these state variables lengthens the code and requires much manual effort. A more efficient technique is to quadruple the phases by using MATLAB® for loops. With twelves phases, the phase assignment for each spacecraft can be conceptualized using Table I.

<table>
<thead>
<tr>
<th></th>
<th>Spacecraft 1</th>
<th>Spacecraft 2</th>
<th>Spacecraft 3</th>
<th>Spacecraft 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coast</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Burn</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Cost</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

7.5.2 Finding the Mesocenter

In order to find the mesocenter of the four spacecraft, the position and velocity vectors of each craft need to be found by converting the final phase of each spacecraft to Cartesian coordinates. This is done using the function mee2rv.m, which is given in Appendix C. Note that because \( L \) is the only state that changes in the final coast phase, all other states are from the end of the burn phase. The position and velocity vectors are of the form given in (19).

\[
\mathbf{r} = [r_x \ r_y \ r_z]
\] (19)
Thus the mesocenter vectors can be found simply using (20), where the superscript in brackets represents the spacecraft number.

\[ \mathbf{r}_{\text{meso}} = \frac{1}{4} \sum_{i=1}^{4} \mathbf{r}^{[i]} \]

\[ \mathbf{v}_{\text{meso}} = \frac{1}{4} \sum_{i=1}^{4} \mathbf{v}^{[i]} \]

7.5.3 Calculating the Side Lengths

There are six sides to the tetrahedron, as illustrated and labeled in Fig. 3. The side lengths are found as the magnitude of the vector difference between the position vectors. An example is given below.

\[ L_{12} = |\mathbf{r}_{12}| = |\mathbf{r}_1 - \mathbf{r}_2| \]

Fig. 3. A schematic of the tetrahedron with the four spacecraft and six edges labeled.

The average side length, which is used to find the volume of a normal tetrahedron, is calculated using (22).

\[ L_{\text{avg}} = \frac{L_{12} + L_{13} + L_{14} + L_{23} + L_{24} + L_{34}}{6} \]
7.5.4 Calculating the Ideal and Actual Volumes

If the tetrahedron were normal, all side lengths would be of the same value. Thus the ideal volume of the formed tetrahedron is as follows:

$$V^* = \frac{L_{\text{avg}}^3}{6\sqrt{2}}$$  \hspace{1cm} (23)

To find the actual volume using the position vectors, (24) is used [19].

$$V_a = \frac{1}{6} |(\mathbf{r}_{12} \times \mathbf{r}_{13}) \cdot \mathbf{r}_{14}|$$  \hspace{1cm} (24)

7.5.5 Finding the Final Semimajor Axis of Each Spacecraft

In order to find the final semimajor axis of each spacecraft, the transformations from the modified equinoctial elements to orbital elements, given in Appendix A, are employed. Thus using the final state of each spacecraft, the final semimajor axis of each spacecraft can be found using the relationship given in (25).

$$a = \frac{p}{1 - p_1^2 - p_2^2}$$  \hspace{1cm} (25)

7.5.6 Setting the Objective

Because GPOPS-II minimizes the objective function, the objective to maximize final mass is achieved by negating the value. For the four spacecraft problem, the objective is to maximize the sum total of the final mass. The objective is conceptualized below, where the superscript in brackets is again the spacecraft number. Note that for one spacecraft, only one mass term is included.

$$\text{objective} = - \left( m_f^{[1]} + m_f^{[2]} + m_f^{[3]} + m_f^{[4]} \right)$$  \hspace{1cm} (26)

7.6 Plotting Functions

To visualize the results, several MATLAB® functions were programmed to generate plots. To plot the trajectory of the one spacecraft solution in Cartesian coordinates, the function plotOneCraft.m is used. Similarly, plotFourCrafts.m is used to plot the trajectories of the four spacecraft solution. The plotEnd.m function is used to plot the endpoints only of the four spacecraft in Cartesian coordinates. Lastly, the function plotControls.m can be used for either the one or four spacecraft solution to plot the three controls versus time. All plotting functions are provided in Appendix G.
7.7 Troubleshooting Techniques

If there is a coding error in the main file, MATLAB® will display an error message before GPOPS-II is executed. The error allows the user to return to the line in the main script and make a correction. If, however, there is a syntax error in the endpoint function, then the error message comes from GPOPS-II and may read as follows:

```
GPOPS-II ERROR: the endpoint function endpointOneCraft can not be evaluated on the initial guess without error
Error in mainOneCraftRev9 (line 247)
output = gpops2(setup);
```

Similarly, a coding error in the continuous function will yield the following GPOPS-II error message:

```
GPOPS-II ERROR: the continuous function continuousOneCraft can not be evaluated on the initial guess without error
Error in mainOneCraftRev9 (line 247)
output = gpops2(setup);
```

As noted, these errors mean there is a problem in the function – the errors do not necessarily mean that there is an error in the initial guess. To find the problem, breakpoints in the functions can be employed [20]. Breakpoints are an effective way to locate and correct the source of the error in the function.
8 RESULTS

A summary of the results obtained at all relevant benchmarks in the problem are presented below.

8.1 ONE SPACECRAFT

The problem was first solved for only one spacecraft. IPOPT was used as the NLP solver. There are necessarily no volumetric, end time, or equal semimajor axis end constraints. The end constraint on the position and velocity vectors of the mesocenter is applied directly to the spacecraft, as can be seen in the code given in Appendix E.

The objective for the one spacecraft solution is \(-0.6735\). Understanding that the objective is negated and scaled, the final mass of the spacecraft is found by unscaling the value as follows:

\[
m_f = |\bar{M} - 0.6735| = 336.75 \text{ kg}
\]

Thus the fuel the spacecraft used is found as

\[
fuel = m_0 - m_f = 163.25 \text{ kg}
\]

The trajectory of the spacecraft is plotted in Fig. 4. Phase two, the burn phase, occurs at the perigee, where the craft is moving at maximum velocity.

---

Fig. 4. The trajectory of the spacecraft is plotted on three axes in Cartesian coordinates.
The controls for the burn phase are plotted in Fig. 5. Note that the tangential control is near maximum for the duration of the burn phase. The radial control is negative before the craft reaches the perigee, zero at the perigee, and positive after the perigee. Finally, the normal control is zero. These results are expected: the tangential thrust is necessary to accelerate the craft to its terminal phase; the radial thrust is used to pass through the desired perigee and continue to the desired orbit; and no normal thrust is necessary since no inclination change is observed.

Fig. 5. The directional controls are plotted against time.

The time spent in each phase, a free variable, is summarized in Table II.

<table>
<thead>
<tr>
<th>Initial Coast Phase (s)</th>
<th>Burn Phase (s)</th>
<th>Final Coast Phase (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1415.6</td>
<td>65.192</td>
<td>5079.1</td>
<td>6559.9</td>
</tr>
</tbody>
</table>

TABLE II
TIME SPENT IN EACH PHASE (ONE SPACECRAFT SOLUTION)
8.2 Four Spacecraft

Building on the one spacecraft solution, end constraints were added gradually. The results are presented briefly for three benchmarks on the way to the final solution: (a) mesocenter and equal final times constrained; (b) mesocenter, equal final times, and side length constrained, and (c) mesocenter, equal final times, side length, and geometric factor constrained. The results for the final solution, which includes the final semimajor axes constraint for repeatability, are presented comprehensively. The equation numbers applicable to each end constraint are specified in the following sections.

The MATLAB® code is given in Appendix F. This code includes all constraints, but can be edited to include less by removing the relevant event constraint(s) from the main and endpoint files.

8.2.1 Mesocenter and Equal Final Times Constrained

This solution includes the end constraints described by (3) and (11). IPOPT was used as the NLP solver, and the objective was determined to be $-2.6943$. The mass of fuel utilized for the spacecraft is determined by first finding the final mass of the spacecraft.

\[ 4m_f = |-2.6943|\bar{M} = 1347.2 \text{ kg} \]  (29)

The fuel utilized by the four spacecraft is determined to be

\[ fuel = 4(m_0 - m_f) = 652.84 \text{ kg} \]  (30)

The average fuel used per spacecraft is 163.21 kg. This is very near to but slightly less than the one spacecraft solution (163.25 kg), which is expected: Since the mesosphere of the four spacecraft is constrained rather than each spacecraft, there is more freedom.

8.2.2 Mesocenter, Equal Final Times, and Side Length Constrained

With the end constraints (3), (11), and (12), the objective was found to be $-2.6942$. The problem was solved using SNOPT. As before, the final mass of the spacecraft and the fuel mass used are found as follows:

\[ 4m_f = |-2.6942|\bar{M} = 1347.1 \text{ kg} \]  (31)

\[ fuel = 4(m_0 - m_f) = 652.90 \text{ kg} \]  (32)

With the side length constraint added, the fuel use increases slightly from the previous solution.

8.2.3 Mesocenter, Equal Final Times, Side Length, and Geometric Factor Constrained

The constraints (3), (11), (12), and (14) are used. The objective was determined to be $-2.6929$, which resulted in a final mass and fuel mass as described below.

\[ 4m_f = |-2.6942|\bar{M} = 1346.4 \text{ kg} \]  (33)
\[ fuel = 4(m_0 - m_f) = 653.55 \text{ kg} \]  

(34)

The addition of the geometric factor constraint led to a more significant increase in the fuel use.

8.2.4 Final Solution (All Constraints Included)

In addition to (3), (11), (12), and (14), the semimajor axis equality constraint (15) is included, resulting in an objective of \(-2.6921\). The problem was solved using SNOPT. This objective yields the following values for final mass and fuel use:

\[ 4m_f = |-2.6921| \bar{M} = 1346.1 \text{ kg} \]  

(35)

\[ fuel = 4(m_0 - m_f) = 653.93 \text{ kg} \]  

(36)

This final result is comparable to the result obtained by [2] of 653.375 kg. The difference in these results is best explained by the stricter side length constraint imposed in this problem.

The trajectory of each spacecraft is of the same form as Fig. 4. The end position of the four crafts is shown in Fig. 6. The tetrahedral shape of this end configuration is visible.

![Diagram of spacecraft end position](image)

**Fig. 6.** The end position of each spacecraft is plotted on three axes in Cartesian coordinates.
The solution for the optimal controls is plotted in Fig. 7. It is noted that these controls vary from the results obtained in [2], although they follow similar patterns. The tangential command is near maximum for each spacecraft. The radial command varies in pairs among the spacecraft: spacecraft 1 and spacecraft 2 have negatively sloping radial command; spacecraft 3 and 4 have positively sloping radial command. The command in the normal direction also varies for each spacecraft. Spacecraft 1 and 2 have near opposite magnitudes of normal control, and accordingly they end on opposite sides of the tetrahedron (see Fig. 6). Spacecraft 3 has nearest to zero normal control, and it ends in the front nearest to the reference orbit. Lastly, spacecraft 4 has the most positive normal control and ends at the top of the tetrahedron.

![Graphs of directional controls for spacecraft 1, 2, 3, and 4.](image)

Fig. 7  The directional controls are plotted against time for each of the four spacecraft.

The time each spacecraft uses in each phase is summarized in Table III. As expected, the time in each phase varies, though the final time is necessarily the same. The burn duration of each spacecraft is very comparable to the results obtained in [2].
**TABLE III**

*Time Spent in Each Phase (Four Spacecraft Solution)*

<table>
<thead>
<tr>
<th></th>
<th>Initial Coast Phase (s)</th>
<th>Burn Phase (s)</th>
<th>Final Coast Phase (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacecraft 1</td>
<td>1412.4</td>
<td>65.296</td>
<td>5083.8</td>
<td>6561.5</td>
</tr>
<tr>
<td>Spacecraft 2</td>
<td>1414.3</td>
<td>65.285</td>
<td>5081.9</td>
<td>6561.5</td>
</tr>
<tr>
<td>Spacecraft 3</td>
<td>1446.4</td>
<td>65.316</td>
<td>5049.8</td>
<td>6561.5</td>
</tr>
<tr>
<td>Spacecraft 4</td>
<td>1437.7</td>
<td>65.282</td>
<td>5058.5</td>
<td>6561.5</td>
</tr>
</tbody>
</table>
9 CONCLUSION

The use of GPOPS-II to find the optimal control solution of the tetrahedral formation problem, adapted from [2], was demonstrated. The problem was first solved using only one spacecraft, resulting in a fuel mass use of 163.25 kg. The problem was expanded gradually to its final form, where the optimal solution resulted in a combined use of 653.93 kg of fuel. The final results were validated by the results obtained in [2]. Furthermore, the methods used to obtain the solutions were described with the intent of aiding other users of GPOPS-II.
REFERENCES


The transformation equations from classical orbital elements to modified equinoctial elements [13] are displayed below. These equations are presented for conceptual aid in understanding the selected boundary conditions (see section, Boundary Conditions). The MATLAB® function for these transformations is provided in Appendix B.

\[ p = a(1 - e^2) \]
\[ P_1 = e \sin(\omega + \Omega) \]
\[ P_2 = e \cos(\omega + \Omega) \]
\[ Q_1 = \tan\left(\frac{i}{2}\right) \sin \Omega \]
\[ Q_2 = \tan\left(\frac{i}{2}\right) \cos \Omega \]
\[ L = i + \omega + \Omega \]
APPENDIX B

The MATLAB® code below is the function used to convert orbital elements to modified equinoctial elements, following the equations in Appendix A, to define the initial conditions.

```matlab
function ee = oe2mee(a,e,in,Om,om,nu)

    p  = a.*(1-e.^2);
    P1 = e.*sin(om+Om);
    P2 = e.*cos(om+Om);
    Q1 = tan(in/2).*sin(Om);
    Q2 = tan(in/2).*cos(0m)
    L  = Om+om+nu;

    ee = [p; P1; P2; Q1; Q2; L];

end
```
APPENDIX C

The MATLAB® code below is used in the endpoint function to convert modified equinoctial elements to Cartesian coordinates (position and velocity vectors).

```matlab
function [r,v] = mee2rv(mee, mu)

p = mee(1);
f = mee(3);
g = mee(2);
h = mee(5);
k = mee(4);
L = mee(6);

q = 1+f.*cos(L)+g.*sin(L);
r = p./q;
alpha2 = h.*h-k.*k;
chi = sqrt(h.*h+k.*k);
s2 = 1+chi.*chi;

X = (r./s2).*(cos(L)+alpha2.*cos(L)+2.*h.*k.*sin(L));
Y = (r./s2).*(sin(L)-alpha2.*sin(L)+2.*h.*k.*cos(L));
Z = (2.*r./s2).*(h.*sin(L)-k.*cos(L));

r = [X Y Z];

VX = -1/s2*sqrt(mu/p)*(sin(L) + alpha2*sin(L) - 2*h*k*cos(L) + g - 2*f*h*k + alpha2*g);
VY = -1/s2*sqrt(mu/p)*(-cos(L) + alpha2*cos(L) + 2*h*k*sin(L) - f + 2*g*h*k + alpha2*f);
VZ = 2/s2*sqrt(mu/p)*(h*cos(L) + k*sin(L) + f*h + g*k);

v = [VX VY VZ];
end
```
APPENDIX D

The MATLAB® function (extractGuesses.m) written to extract the guesses from a previous solution of the problem with either one spacecraft (three phases) or four spacecraft (twelve phases) is shown below.

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
% extractGuesses.m %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %

% Extract results from previous solution to generate guesses. The first % input ("string") is the file name of the solution to be loaded (ex: % 'Rev1.mat'). The second input ("phaseCount") is the number of phases in % the solution being loaded (ex: 12).

function prevSol = extractGuesses(string,phaseCount)

%% Load Results
load(string)
nCount = phaseCount/3;

%% Organize Results

% Preallocate structures
state   = cell(1,phaseCount);
time    = cell(1,phaseCount);
control = cell(1,phaseCount);

for iphase = 1:phaseCount
    state{iphase}   = output.result.solution.phase(iphase).state;
time{iphase}    = output.result.solution.phase(iphase).time;
    control{iphase} = output.result.solution.phase(iphase).control;
end

%% Extract First Phase

for n = 1:nCount

    iphase = 3*(n-1)+1;

    % Final Conditions
    prevSol.state.Lf(iphase)   = state{iphase}(end);
    prevSol.time.tf(iphase)    = time{iphase}(end);

end

%% Extract Phase 2
```

for n = 1:nCount

    iphase = 3*(n-1)+2;

    % Initial Conditions
    prevSol.state.p0(iphase) = state{iphase}(1,1);
    prevSol.state.F10(iphase) = state{iphase}(1,2);
    prevSol.state.F20(iphase) = state{iphase}(1,3);
    prevSol.state.Q10(iphase) = state{iphase}(1,4);
    prevSol.state.Q20(iphase) = state{iphase}(1,5);
    prevSol.state.L0(iphase) = state{iphase}(1,6);
    prevSol.state.m0(iphase) = state{iphase}(1,7);
    prevSol.control.ur0(iphase) = control{iphase}(1,1);
    prevSol.control.ut0(iphase) = control{iphase}(1,2);
    prevSol.control.un0(iphase) = control{iphase}(1,3);
    prevSol.time.t0(iphase) = time{iphase}(1);

    % Final Conditions
    prevSol.state.pf(iphase) = state{iphase}(end,1);
    prevSol.state.F1f(iphase) = state{iphase}(end,2);
    prevSol.state.F2f(iphase) = state{iphase}(end,3);
    prevSol.state.Q1f(iphase) = state{iphase}(end,4);
    prevSol.state.Q2f(iphase) = state{iphase}(end,5);
    prevSol.state.Lf(iphase) = state{iphase}(end,6);
    prevSol.state.mf(iphase) = state{iphase}(end,7);
    prevSol.control.urf(iphase) = control{iphase}(end,1);
    prevSol.control.utf(iphase) = control{iphase}(end,2);
    prevSol.control.unf(iphase) = control{iphase}(end,3);
    prevSol.time.tf(iphase) = time{iphase}(end);

end

%% Extract Phase 3

for n = 1:nCount

    iphase = 3*(n-1)+3;

    % Initial Conditions
    prevSol.state.L0(iphase) = state{iphase}(1);
    prevSol.time.t0(iphase) = time{iphase}(1);

    % Final Conditions
    prevSol.state.Lf(iphase) = state{iphase}(end);
    prevSol.time.tf(iphase) = time{iphase}(end);

end

%% Parameters

if phaseCount == 3;
    prevSol.param.p = output.result.solution.parameter(1);
end
prevSol.param.P1 = output.result.solution.parameter(2);
prevSol.param.P2 = output.result.solution.parameter(3);
end

if phaseCount == 12;
    for n = 1:phaseCount
        prevSol.param.value(n) = output.result.solution.parameter(n);
    end
end

if and(phaseCount ~= 3, phaseCount ~= 12)
    error('phaseCount can only be 3 or 12.')</nend
end
APPENDIX E

The main script, endpoint function, and continuous function for the problem coded with only one spacecraft is given below. With only one spacecraft, no tetrahedron shape constraints, repeatability constraints, or equal end time constraints were necessary. Further, the position vector and velocity vector constraints are applied directly to the spacecraft, since there is no mesocenter.

```
%%%%% mainOneCraft.m

% Problem is based off of the paper "Optimal Configuration of Tetrahedral Spacecraft Formations."
% This script is the starting point to solve the final problem. One spacecraft is used. The craft starts in a circular orbit and ascends to the apogee of an orbit of defined shape. There are three phases: coast, burn, coast.

close all; clear all;

%% Provide All Physical Data for Problem

% Orbital Constants
mu = 3.986004415E14;            % (m^3/s^2) standard gravitational parameter of Earth
Re = 6378145;                    % (m) radius of Earth
g0 = 9.80665;                    % (m/s^2) acceleration due to gravity

% Orbital Parameters
h0 = 600*1e3;                    % (m) initial altitude
a0 = Re+h0;                      % (m) initial semiminor axis (and apogee and perigee radius)
hf = 7000*1e3;                   % (m) final altitude
Ra = Re+hf;                      % (m) final apogee radius
af = (Ra+a0)/2;                  % (m) final semimajor axis
ef = (Ra-a0)/(Ra+a0);            % (no units) final eccentricity

% Spacecraft Parameters
T = 7015;                        % (N) constant maximum thrust
Isp = 285.7;                     % (s) specific impulse
m0 = 500;                        % (kg) initial mass of one spacecraft
mDry = 200;                      % (kg) dry Wmass of one spacecraft

% Conversions
RADIAN = pi/180;
DEGREE = 180/pi;
auxdata.RADIAN = RADIAN;
auxdata.DEGREE = DEGREE;
```
%% Manually Scale the Problem
% From this point, all variables are scaled. Data added to auxdata for use
% in other functions.

LBar   = Re;   % (m)
TBar   = sqrt(Re^3/mu);   % (s) 801.8126 s, for reference
MBar   = m0;   % (kg)
VBar   = LBar/TBar;   % (m/s)
ABar   = VBar/TBar;   % (m/s^2)
FBar   = ABar*MBar;   % (N)
MuBar  = LBar^3/TBar^2;   % (m^3/s^2)

% Orbital Constants
auxdata.SCALEDRe   = Re/LBar;
auxdata.SCALEDmu   = mu/MuBar;
auxdata.SCALEDg0   = g0/ABar;

% Scaled Orbital Parameters
auxdata.SCALEDh0    = h0/LBar;
auxdata.SCALEDa0    = a0/LBar;
auxdata.SCALEDf    = hf/LBar;
auxdata.SCALEDRa    = Ra/LBar;
auxdata.SCALEDaf    = af/LBar;   % 1.5958, for reference
auxdata.SCALEDef    = ef;   % 0.3144, for reference

% Spacecraft Parameters
auxdata.SCALEDT     = T/FBar;
auxdata.SCALEDIsp   = Isp/TBar;
auxdata.SCALEDm0    = m0/MBar;
auxdata.SCALEDmDry  = mDry/MBar;

% Initial Conditions (IC)
% IC's are defined in orbital elements (OE). The dynamics, and therefore
% the states, are in modified equinoctial elements (MEE).

% Define IC's in OE
e0  = 0;   % (rad) eccentricity
in0 = 28*RADIAN;   % (rad) inclination (should stay roughly constant)
Om0 = 0;   % (rad) longitude of ascending node
om0 = 270*RADIAN;   % (rad) argument of perigee (expected to stay roughly
constant)
nu0 = 270*RADIAN;   % (rad) true anomaly (initial location arbitrary)

% Convert IC's from OE to MEE
ee0 = oe2mee(auxdata.SCALEDa0, e0, in0, Om0, om0, nu0);
p0  = ee0(1);
P10 = ee0(2);
P20 = ee0(3);
Q10 = ee0(4);
Q20 = ee0(5);
L0  = ee0(6);

% Added to auxdata for use in plotting
auxdata.p0  = p0;   % 1.0941, for reference
auxdata.P10 = P10; % 0, for reference
auxdata.P20 = P20; % 0, for reference
auxdata.Q10 = Q10; % 0, for reference
auxdata.Q20 = Q20; % 0.2493, for reference
auxdata.L0  = L0; % 9.4248, for reference

%% Mid Conditions
% Mins and maxes derived from conversion between OE and MEE.

pMin    = 0.5*auxdata.SCALEDa0;
pMax    = 1.5*auxdata.SCALEDaf;
P1Min   = -1; % -eMax
P1Max   = 1; % eMax
P2Min   = -1; % -eMax
P2Max   = 1; % eMax
Q1Min   = -1; % -tan(iMax/2)
Q1Max   = 1; % tan(iMax/2)
Q2Min   = -1; % -tan(iMax/2)
Q2Max   = 1; % tan(iMax/2)
LMin    = -10*pi;
LMax    = 10*pi;

StateMin = [pMin P1Min P2Min Q1Min Q2Min LMin auxdata.SCALEDmDry];
StateMax = [pMax P1Max P2Max Q1Max Q2Max LMax auxdata.SCALEDm0];

%% Final Conditions
% The final eccentricity, semimajor axis, position, and inclination are
% fixed. They are constrained as events in the endpoint function by
% constraining the position and velocity in Cartesian coordinates.

%% Time Constraints
t0   = 0;
tMax = 10E5/TBar; % arbitrary maximum time

%% Set Up Bounds and Guesses for Each Phase

% Extract guesses from previous solution
prevSol = extractGuess('output.mat',3);

% Phase 1: Coast

iphas e = 1;

bounds.phase(iphase).initialtime.lower  = t0;
bounds.phase(iphase).initialtime.upper  = t0;
bounds.phase(iphase).finaltime.lower    = t0;
bounds.phase(iphase).finaltime.upper    = tMax;
bounds.phase(iphase).initialstate.lower = L0;
bounds.phase(iphase).initialstate.upper = L0;
bounds.phase(iphase).state.lower        = LMin;
bounds.phase(iphase).state.upper        = LMax;
bounds.phase(iphase).finalstate.lower   = LMin;
bounds.phase(iphase).finalstate.upper   = LMax;
% Phase 2: Burn

iphase = 2;

bounds.phase(iphase).initialtime.lower  = t0;
bounds.phase(iphase).initialtime.upper  = tMax;
bounds.phase(iphase).finaltime.lower    = t0;
bounds.phase(iphase).finaltime.upper    = tMax;
bounds.phase(iphase).initialstate.lower = [p0 P10 P20 Q10 Q20 LMin auxdata.SCALEDm0];
bounds.phase(iphase).initialstate.upper = [p0 P10 P20 Q10 Q20 LMax auxdata.SCALEDm0];
bounds.phase(iphase).state.lower        = StateMin;
bounds.phase(iphase).state.upper        = StateMax;
bounds.phase(iphase).finalstate.lower   = StateMin;
bounds.phase(iphase).finalstate.upper   = StateMax;
bounds.phase(iphase).control.lower      = -1*ones(1,3);
bounds.phase(iphase).control.upper      = ones(1,3);
bounds.phase(iphase).path.lower         = 1;
bounds.phase(iphase).path.upper         = 1;

% guess.phase(iphase).control(:,1)    = [prevSol.control.ur0(iphase); 
% prevSol.control.urf(iphase)];
% guess.phase(iphase).control(:,2)    = [prevSol.control.ut0(iphase); 
% prevSol.control.utf(iphase)];
% guess.phase(iphase).control(:,3)    = [prevSol.control.un0(iphase); 
% prevSol.control.unf(iphase)];
guess.phase(iphase).control(:,1)    = [0; 0];
guess.phase(iphase).control(:,2)    = [1; 1];
guess.phase(iphase).control(:,3)    = [0; 0];

% Phase 3: Coast

iphase = 3;

bounds.phase(iphase).initialtime.lower  = t0;
bounds.phase(iphase).initialtime.upper  = tMax;
bounds.phase(iphase).finaltime.lower    = t0;
bounds.phase(iphase).finaltime.upper    = tMax;
bounds.phase(iphase).initialstate.lower = LMin;
bounds.phase(iphase).initialstate.upper = LMax;
bounds.phase(iphase).state.lower = LMin;
bounds.phase(iphase).state.upper = LMax;
bounds.phase(iphase).finalstate.lower = LMin;
bounds.phase(iphase).finalstate.upper = LMax;

guess.phase(iphase).time = [prevSol.time.t0(iphase);
prevSol.time.tf(iphase)];

%% Set Up Parameters

bounds.parameter.lower  = [pMin P1Min P2Min];
bounds.parameter.upper  = [pMax P1Max P2Max];
guess.parameter        = [prevSol.param.p prevSol.param.P1
prevSol.param.P2];

%% Set up Event Constraints That Link Phases

bounds.eventgroup(1).lower = zeros(1,2);
bounds.eventgroup(1).upper = zeros(1,2);
bounds.eventgroup(2).lower = zeros(1,5);
bounds.eventgroup(2).upper = zeros(1,5);
bounds.eventgroup(3).lower = zeros(1,6);
bounds.eventgroup(3).upper = zeros(1,6);

%% Provide an Initial Mesh in Each Phase

meshphase = struct();
phaseCount = 3;
for iphase=1:phaseCount
    meshphase(iphase).colpoints = 4*ones(1,10);
    meshphase(iphase).fraction = 0.1*ones(1,10);
end

%% Assemble All Information into Setup Structure

setup.name = 'Tetrahedral Spacecraft Formation';
setup.functions.continuous = @continuousOneCraft;
setup.functions.endpoint = @endpointOneCraft;
setup.mesh.phase = meshphase;
setup.nlp.solver = 'ipopt';
setup.nlp.ipoptoptions.tolerance = 1e-12; % 1e-7 minimum for good accuracy
setup.bounds = bounds;
setup.guess = guess;
setup.auxdata = auxdata;
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'sparseNaN';
setup.mesh.method = 'hp-PattersonRao';
setup.mesh.tolerance = 1e-7; % 1e-3 minimum for good accuracy
%setup.mesh.maxiteration = 20;
setup.method = 'RPM-Differentiation';
function output = endpointOneCraft(input)

% Define Constants and Variables from Input
mu = input.auxdata.SCALEDmu;
p = input.parameter(1);
P1 = input.parameter(2);
P2 = input.parameter(3);
params = [p P1 P2];
RADIAN = pi/180;

phaseCount = 3;
t0 = cell(1,phaseCount);
tf = cell(1,phaseCount);
x0 = cell(1,phaseCount);
xf = cell(1,phaseCount);
for phaseIter = 1:phaseCount
    t0{phaseIter} = input.phase(phaseIter).initialtime;
tf{phaseIter} = input.phase(phaseIter).finaltime;
x0{phaseIter} = input.phase(phaseIter).initialstate;
    xf{phaseIter} = input.phase(phaseIter).finalstate;
end

% Event Constraints
% Linkage Between Phases 1 and 2
output.eventgroup(1).event = [x0{2}(6) - xf{1}, t0{2} - tf{1}];

% Linkage Between Phases 2 and 3
output.eventgroup(2).event = [params - xf{2}(1:3), x0{3} - xf{2}(6), t0{3} - tf{2}];

% Final Conditions

% Final position and velocity
finalState = [xf{2}(1),xf{2}(2),xf{2}(3),xf{2}(4),xf{2}(5),xf{3}(1)];

[rr,vv] = mee2rv(finalState,mu);

rMeso = [rr(1) rr(2) rr(3)];
vMeso = [vv(1) vv(2) vv(3)];
inf = 28*RADIAN;
Omf = 0;
omf = 270*RADIAN;
nuf = 180*RADIAN;
af = input.auxdata.SCALEDaf;
ef = input.auxdata.SCALEDef;
oe = [af ef inf Omf omf nuf];
[riApo, viApo] = oe2rv(oe,mu);

output.eventgroup(3).event = [rMeso - riApo' vMeso - viApo'];

output.objective = -xf(2)(7);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% continuousOneCraft.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function output = continuousOneCraftRev9(input)

%% Define Constants from Input

mu  = input.auxdata.SCALEDmu;
g0  = input.auxdata.SCALEDg0;
T   = input.auxdata.SCALEDT;
Isp = input.auxdata.SCALEDIsp;

%% Coast Phase
% The first and third phases are the coast phases. There are three parameters
% (p, P1, and P2), but they are known from initial conditions. There is one
% state (L). There are no controls.

output = struct(); % preallocate for speed
iphase = 1

% define parameters and states
p  = input.auxdata.p0;
P1 = input.auxdata.P10;
P2 = input.auxdata.P20;
L  = input.phase(iphase).state(:,1);

% dynamics of the coarse phase
Ldot = sqrt(mu*p)./p.^2.*(1+P1.*sin(L)+P2.*cos(L)).^2;

output(iphase).dynamics = Ldot;

%% Burn Phase
% The second phase is the burn phase. There are no parameters, seven
% states (p, P1, P2, Q1, Q2, L, and m), and three controls (ur, ut, and
% un).

iphase = 2
% define states and controls (time included for size reference for
% mdot - see below)
t = input.phase(iphase).time;
p = input.phase(iphase).state(:,1);
P1 = input.phase(iphase).state(:,2);
P2 = input.phase(iphase).state(:,3);
Q1 = input.phase(iphase).state(:,4);
Q2 = input.phase(iphase).state(:,5);
L = input.phase(iphase).state(:,6);
m = input.phase(iphase).state(:,7);
ur = input.phase(iphase).control(:,1);
ut = input.phase(iphase).control(:,2);
un = input.phase(iphase).control(:,3);

% path definition
path = ur.^2 + ut.^2 + un.^2;

% dynamics of the burn phase
w = 1+P1.*sin(L)+P2.*cos(L);
sSquare = 1+Q1.^2+Q2.^2;
ar = T./m.*ur;
at = T./m.*ut;
an = T./m.*un;
pdot = 2*p./w.*sqrt(p./mu).*at;
P1dot = sqrt(p./mu).*((-ar.*cos(L)+((w+1).*sin(L)+P1).*at./w+(Q2.*sin(L)-Q1.*cos(L)).*P2.*an./w);
P2dot = sqrt(p./mu).*((ar.*sin(L)+((w+1).*cos(L)+P2).*at./w-(Q2.*sin(L)-Q1.*cos(L))).*P1.*an./w);
Q1dot = sqrt(p./mu).*((sSquare./2.*w).*an.*sin(L);
Q2dot = sqrt(p./mu).*((sSquare./2.*w).*an.*cos(L);
LdotT = sqrt(mu.*p).*(w./p).^2+sqrt(p./mu).*(Q2.*sin(L)-Q1.*cos(L)).*an./w;

mdot = -T/(g0*Isp)*ones(size(t));

output(iphase).dynamics = [pdot,P1dot,P2dot,Q1dot,Q2dot,LdotT,mdot];
output(iphase).path = path;

%% Coast Phase
% There are three parameters (p, P1, and P2), one state (L), and no controls.

iphase = 3

p = input.phase(iphase).parameter(:,1);
P1 = input.phase(iphase).parameter(:,2);
P2 = input.phase(iphase).parameter(:,3);
L = input.phase(iphase).state(:,1);

% dynamics of the coase phase
Ldot = sqrt(mu*p)./p.^2.*(1+P1.*sin(L)+P2.*cos(L)).^2;

output(iphase).dynamics = Ldot;

end
The MATLAB® code for the four craft problem, including all final constraints, is provided below.

```matlab
%% Tetrahedral Formation Problem
%% Problem is based off of the paper "Optimal Configuration of Tetrahedral
%% Spacecraft Formations."
%% Four crafts start in a circular orbit and ascend to the apogee of an
%% orbit of defined shape. There are three phases for each craft: coast,
%% burn, coast.

close all; clear all;

%% Provide All Physical Data for Problem

% Orbital Constants
mu = 3.986004415E14; ▶ (m^3/s^2) standard gravitational parameter of
Earth
Re = 6378145; ▶ (m) radius of Earth
g0 = 9.80665; ▶ (m/s^2) acceleration due to gravity

% Orbital Parameters
h0 = 600*1e3; ▶ (m) initial altitude
a0 = Re+h0; ▶ (m) initial semiminor axis (and apogee and
perigee radius)
hf = 7000*1e3; ▶ (m) final altitude
Ra = Re+hf; ▶ (m) final apogee radius
af = (Ra+a0)/2; ▶ (m) final semimajor axis
aMin = a0 - 200*1e3; ▶ (m) minimum semimajor axis (for bounds)
aMax = af + 400e3; ▶ (m) maximum semimajor axis (for bounds)
ef = (Ra-a0)/(Ra+a0); ▶ (no units) final eccentricity

% Spacecraft Parameters
T = 7015; ▶ (N) constant maximum thrust
Isp = 285.7; ▶ (s) specific impulse
m0 = 500; ▶ (kg) initial mass of one spacecraft
mDry = 200; ▶ (kg) dry Wmass of one spacecraft

% Conversions
RADIAN = pi/180;
DEGREE = 180/pi;
auxdata.RADIAN = RADIAN;
auxdata.DEGREE = DEGREE;

%% Manually Scale the Problem
%% From this point, all variables are scaled. Data added to auxdata for use
%% in other functions.
```
LBar = Re; % (m)
TBar = sqrt(Re^3/mu); % (s) 801.8126 s, for reference
MBar = m0; % (kg)
VBar = LBar/TBar; % (m/s)
ABar = VBar/TBar; % (m/s^2)
FBar = ABar*MBar; % (N)
MuBar = LBar^3/TBar^2; % (m^3/s^2)

% Orbital Constants
auxdata.SCALEDRe = Re/LBar;
auxdata.SCALEDmu = mu/MuBar;
auxdata.SCALEDg0 = g0/ABar;

% Scaled Orbital Parameters
auxdata.SCALEDh0 = h0/LBar;
auxdata.SCALEDa0 = a0/LBar;
auxdata.SCALEDhf = hf/LBar;
auxdata.SCALEDRa = Ra/LBar;
auxdata.SCALEDaf = af/LBar; % 1.5958, for reference
auxdata.SCALEDaMin = aMin/LBar;
auxdata.SCALEDaMax = aMax/LBar;
auxdata.ef = ef; % 0.3144, for reference

% Spacecraft Parameters
auxdata.SCALEDT = T/FBar;
auxdata.SCALEDIsp = Isp/TBar;
auxdata.SCALEDm0 = m0/MBar;
auxdata.SCALEDmDry = mDry/MBar;

% Initial Conditions (IC)
% IC's are defined in orbital elements (OE). The dynamics, and therefore
% the states, are in modified equinoctial elements (MEE).

% Define IC's in OE
e0 = 0; % (rad) eccentricity
in0 = 28*RADIAN; % (rad) inclination (should stay roughly constant)
Om0 = 0; % (rad) longitude of ascending node
om0 = 270*RADIAN; % (rad) argument of perigee (expected to stay roughly
c  onstant)
nu0 = 270*RADIAN; % (rad) true anomaly (initial location arbitrary)

% Convert IC's from OE to MEE
ee0 = oe2mee(auxdata.SCALEDa0, e0, in0, Om0, om0, nu0);
p0 = ee0(1);
P10 = ee0(2);
P20 = ee0(3);
Q10 = ee0(4);
Q20 = ee0(5);
L0 = ee0(6);

% Added to auxdata for use in plotting
auxdata.p0 = p0; % 1.0941, for reference
auxdata.P10 = P10; % 0, for reference
auxdata.P20 = P20;  % 0, for reference
auxdata.Q10 = Q10;  % 0, for reference
auxdata.Q20 = Q20;  % 0.2493, for reference
auxdata.L0 = L0;  % 9.4248, for reference

%% Mid Conditions
% Mins and maxes derived from conversion between OE and MEE.

pMin = 0.1*auxdata.SCALEDa0;  % CHECK VALUE AGAINST BEGUM'S
pMax = 2*auxdata.SCALEDaf;
P1Min = -1;  % -eMax
P1Max = 1;  % eMax
P2Min = -1;  % -eMax
P2Max = 1;  % eMax
Q1Min = -1;  % -tan(iMax/2)
Q1Max = 1;  % tan(iMax/2)
Q2Min = -1;  % -tan(iMax/2)
Q2Max = 1;  % tan(iMax/2)
LMin = -10*pi;
LMax = 10*pi;

StateMin = [pMin P1Min P2Min Q1Min Q2Min LMin auxdata.SCALEDmDry];
StateMax = [pMax P1Max P2Max Q1Max Q2Max LMax auxdata.SCALEDm0];

%% Final Conditions
% These final conditions are implemented in the endpoint function by
% constraining the final position and velocity vectors (Cartesian
% coordinates). The final semimajor axis and eccentricity are defined in
% Orbital Parameters section above.
auxdata.inf = in0;
auxdata.Omf = Om0;
auxdata.omf = om0;
auxdata.nuf = 180*RADIAN;  % Ends at apogee

%% Time Constraints
t0 = 0;
tMax = 10E5/TBar;  % arbitrary maximum time

%% Set Up Bounds and Guesses for Each Phase
% Extract guesses from previous solution
prevSol = extractGuess('output.mat',12);

% Phase 1: Coast
for n = 1:4
    iphase = 3*(n-1)+1;

    bounds.phase(iphase).initialtime.lower = t0;
    bounds.phase(iphase).initialtime.upper = t0;
    bounds.phase(iphase).finaltime.lower = t0;
    bounds.phase(iphase).finaltime.upper = tMax;
bounds.phase(iphase).initialstate.lower = L0;
bounds.phase(iphase).initialstate.upper = L0;
bounds.phase(iphase).state.lower = LMin;
bounds.phase(iphase).state.upper = LMax;
bounds.phase(iphase).finalstate.lower = LMin;
bounds.phase(iphase).finalstate.upper = LMax;

guess.phase(iphase).time = [t0; prevSol.time.tf(iphase)];
guess.phase(iphase).state(:,1) = [L0; prevSol.state.Lf(iphase)];

end

% Phase 2: Burn

for n = 1:4

    iphase = 3*(n-1)+2;

    bounds.phase(iphase).initialtime.lower = t0;
    bounds.phase(iphase).initialtime.upper = tMax;
    bounds.phase(iphase).finaltime.lower = t0;
    bounds.phase(iphase).finaltime.upper = tMax;
    bounds.phase(iphase).initialstate.lower = [p0 P10 P20 Q10 Q20 LMin auxdata.SCALEDm0];
    bounds.phase(iphase).initialstate.upper = [p0 P10 P20 Q10 Q20 LMax auxdata.SCALEDm0];
    bounds.phase(iphase).state.lower = StateMin;
    bounds.phase(iphase).state.upper = StateMax;
    bounds.phase(iphase).finalstate.lower = StateMin;
    bounds.phase(iphase).finalstate.upper = StateMax;
    bounds.phase(iphase).control.lower = -1*ones(1,3);
    bounds.phase(iphase).control.upper = ones(1,3);
    bounds.phase(iphase).path.lower = 1;
    bounds.phase(iphase).path.upper = 1;

    guess.phase(iphase).time = [prevSol.time.t0(iphase);
prevSol.time.tf(iphase)];
    guess.phase(iphase).state(:,1) = [p0; prevSol.state.pf(iphase)];
    guess.phase(iphase).state(:,2) = [P10; prevSol.state.P1f(iphase)];
    guess.phase(iphase).state(:,3) = [P20; prevSol.state.P2f(iphase)];
    guess.phase(iphase).state(:,4) = [Q10; prevSol.state.Q1f(iphase)];
    guess.phase(iphase).state(:,5) = [Q20; prevSol.state.Q2f(iphase)];
    guess.phase(iphase).state(:,6) = [prevSol.state.L0(iphase);
prevSol.state.Lf(iphase)];
    guess.phase(iphase).state(:,7) = [auxdata.SCALEDm0;
prevSol.state.mf(iphase)];
    guess.phase(iphase).control(:,1) = [prevSol.control.ur0(iphase);
prevSol.control.urf(iphase)];
    guess.phase(iphase).control(:,2) = [prevSol.control.ut0(iphase);
prevSol.control.utf(iphase)];
    guess.phase(iphase).control(:,3) = [prevSol.control.un0(iphase);
prevSol.control.unf(iphase)];

end
% Phase 3: Coast

for n = 1:4

    iphase = 3*(n-1)+3;

    bounds.phase(iphase).initialtime.lower = t0;
    bounds.phase(iphase).initialtime.upper = tMax;
    bounds.phase(iphase).finaltime.lower = t0;
    bounds.phase(iphase).finaltime.upper = tMax;
    bounds.phase(iphase).initialstate.lower = LMin;
    bounds.phase(iphase).initialstate.upper = LMax;
    bounds.phase(iphase).state.lower = LMin;
    bounds.phase(iphase).state.upper = LMax;
    bounds.phase(iphase).finalstate.lower = LMin;
    bounds.phase(iphase).finalstate.upper = LMax;

    guess.phase(iphase).time = [prevSol.time.t0(iphase);
    prevSol.time.tf(iphase)];
    guess.phase(iphase).state(:,1) = [prevSol.state.L0(iphase);
    prevSol.state.Lf(iphase)];
end

%% Set Up Parameters

lowerParam = [pMin P1Min P2Min];
upperParam = [pMax P1Max P2Max];

bounds.parameter.lower = [lowerParam lowerParam lowerParam lowerParam];
bounds.parameter.upper = [upperParam upperParam upperParam upperParam];
guess.parameter = prevSol.param.value(1:12);

%% Set up Event Constraints That Link Phases

% Linkage from first to second phase of each craft
for n = 1:4
    bounds.eventgroup(n).lower = zeros(1,2);
    bounds.eventgroup(n).upper = zeros(1,2);
end

% Linkage from second to third phase of each craft
for n = 5:8
    bounds.eventgroup(n).lower = zeros(1,5);
    bounds.eventgroup(n).upper = zeros(1,5);
end

% Position and velocity vector constraints
% The final position and velocity of the mesocenter (average) of the four
% crafts are constrained.
bounds.eventgroup(9).lower = zeros(1,6);
bounds.eventgroup(9).upper = zeros(1,6);
% Final time constraint
% All the spacecraft must end their last phase at the same time.
bounds.eventgroup(10).lower = zeros(1,3);
bounds.eventgroup(10).upper = zeros(1,3);

% L constraint
% Average length between the spacecraft must be between 4000 and 18000 m.
bounds.eventgroup(11).lower = 4000/LBar*ones(1,6);
bounds.eventgroup(11).upper = 18000/LBar*ones(1,6);

% Volume constraint
% The spacecrafts must form a tetrahedron of performance criteria defined
% by the Geometric Factor (QR8). Values of QR8 greater than or equal to 0.9
% are accepted. A value of QR8 higher than 1 is not expected.
bounds.eventgroup(12).lower = 0.9;
bounds.eventgroup(12).upper = 1.1;

% Semimajor axis constraint
% Each craft must have the same semimajor axis in order to have the same
% period.
bounds.eventgroup(13).lower = zeros(1,3);
bounds.eventgroup(13).upper = zeros(1,3);

%% Provide an Initial Mesh in Each Phase
meshphase = struct();
phaseCount = 12;
for iphase=1:phaseCount
    meshphase(iphase).colpoints = 4*ones(1,10);
    meshphase(iphase).fraction  = 0.1*ones(1,10);
end

%% Assemble All Information into Setup Structure
setup.name = 'Tetrahedral Spacecraft Formation';
setup.functions.continuous = @continuousFourCraft;
setup.functions.endpoint = @endpointFourCraft;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;
setup.mesh.phase = meshphase;
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'sparseNaN';
setup.mesh.method = 'hp-PattersonRao';
setup.mesh.tolerance = 1e-5;
setup.mesh.maxiterations = 20;
setup.method = 'RPM-Differentiation';
setup.nlp.solver = 'snopt';

%% Solve Problem using GPOPS2
output = gops2(setup);
function output = endpointFourCraft(input)
phaseCount = 12;

% Define Constants and Variables from Input

% Constants
mu = input.auxdata.SCALEDmu;

% Time and States

t0 = cell(1,phaseCount);  % preallocate for speed
tf = cell(1,phaseCount);
x0 = cell(1,phaseCount);
xf = cell(1,phaseCount);
for iphase = 1:phaseCount
    t0{iphase} = input.phase(iphase).initialtime;
    tf{iphase} = input.phase(iphase).finaltime;
    x0{iphase} = input.phase(iphase).initialstate;
    xf{iphase} = input.phase(iphase).finalstate;
end

% Parameters
% Row vector "params" carries p, P1, and P2 for each craft as follows:
% Craft 1: params(1:3)
% Craft 2: params(4:6)
% Craft 3: params(7:9)
% Craft 4: params(10:12)
params = input.parameter(1:12);

% Event Contraints

% Linkage Between First and Second Phase of Each Craft
% Second phases: 2, 5, 8, 11
% Event constraints: 1, 2, 3, 4
for n = 1:4
    iphase = 3*(n-1)+2;
    output.eventgroup(n).event = [x0{iphase}(6)-xf{iphase-1}, ...
                                t0{iphase} - tf{iphase-1}];
end

% Linkage Between Second and Third Phase of Each Craft
% Third phases: 3, 6, 9, 12
% Event constraints: 5, 6, 7, 8
for n = 1:4
    iphase = 3*(n-1)+3;
    output.eventgroup(n+4).event = [params(3*(n-1)+1:3*(n-1)+3)...
                               - xf{iphase-1}{1:3}, x0{iphase} - xf{iphase-1}{6}, t0{iphase}...
                               - tf{iphase-1}];
end
% Final Conditions
% The final position and velocity of the mesocenter (average) of the four
% crafts are constrained. The final state is final p, P1, P2, Q1, and Q2
% from the second phase and final L from the third phase.
% Event constraint: 9

% Final position and velocity of each craft
r = cell(1,4); % preallocate for speed
v = cell(1,4);
for n = 1:4
    ighase = 3*(n-1)+2;
    finalState = [xf{ighase}(1) xf{ighase}(2) xf{ighase}(3) ...
    xf{ighase}(4) xf{ighase}(5) xf{ighase+1}(1)];
    [rr,vv] = mee2rv(finalState,mu);
    r{n} = rr;
    v{n} = vv;
end

% Find position mesocenter
rSum = r{1}+r{2}+r{3}+r{4};
rMeso = 1/4*rSum;

% Find velocity mesocenter
vSum = v{1}+v{2}+v{3}+v{4};
vMeso = 1/4*vSum;

% Final conditions at the apogee
af = input.auxdata.SCALEDaf;
ef = input.auxdata.ef;
inf = input.auxdata.inf;
Omf = input.auxdata.Omf;
omf = input.auxdata.omf;
nuf = input.auxdata.nuf;
OE = [af ef inf Omf omf nuf];
[riApo, viApo] = oe2rv(OE,mu); % riApo and viApo are column vectors

output.eventgroup(9).event = [rMeso - riApo' vMeso - viApo'];

% Final time constraint
output.eventgroup(10).event = [tf{12}-tf{9} tf{9}-tf{6} tf{6}-tf{3}];

% L constraint
% Event constraint: 12
L12 = (r{1}-r{2});
magL12 = sqrt(L12*L12');

L13 = (r{1}-r{3});
magL13 = sqrt(L13*L13');

L14 = (r{1}-r{4});
magL14 = sqrt(L14*L14');

L23 = (r{2}-r{3});
magL23 = sqrt(L23*L23');

L24 = (r(2)-r(4));
magL24 = sqrt(L24*L24');

L34 = (r(3)-r(4));
magL34 = sqrt(L34*L34');

L = [magL12 magL13 magL14 magL23 magL24 magL34];

output.eventgroup(11).event = L;

% Volume constraint
% Event constraint: 12

% Volume of a tetrahedron: V = 1/6*abs((vec12 X vec13) * vec14)
vec12 = r(2)-r(1);
vec13 = r(3)-r(1);
vec14 = r(4)-r(1);
vecCross  = cross(vec12,vec13);
scalarDot = dot(vecCross,vec14);
V = 1/6*abs(scalarDot);
% V = 1/6*det([vec12; vec13; vec14]);

% Volume of a regular tetrahedron
Lavg = mean(L);
Vreg = Lavg^3/(6*sqrt(2));

% Geometric Factor (QR8)
QR8 = V/Vreg;

output.eventgroup(12).event = QR8;

% Final semimajor axis constraint
% Semimajor axis constraint
% Event constraint: 13

% Find semimajor axis of each craft
for n = 1:4
    iphase = 3*(n-1)+2;
    ef = sqrt(xf{iphase}(2)^2+xf{iphase}(3)^2); % From mee2oe, ef = sqrt(P1^2+P2^2)
    af(n) = xf{iphase}(1)/(1-ef^2); % From mee2oe, af = p/(1-P1^2-P2^2)
end

% Three linkages to confirm all semimajor axes are the same
afLinkage = zeros(1,3); % preallocate for speed
for n = 1:3
    afLinkage(n) = af(n+1)-af(n);
end
output.eventgroup(13).event = afLinkage;

%%% Objective

massSum = xf{2}(7)+xf{5}(7)+xf{8}(7)+xf{11}(7);
output.objective = -massSum;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% continuousFourCraft.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function output = continuousFourCraft(input)
output = struct(); % preallocate for speed

%%% Define Constants from Input

mu  = input.auxdata.SCALEDmu;
g0  = input.auxdata.SCALEDg0;
T   = input.auxdata.SCALEDT;
Isp = input.auxdata.SCALEDIsp;

%%% Coast Phase
% The first and last phases are the coast phases for each craft. There are
% three constants (p, P1, and P2) and one state (L). There are no controls.

% Craft 1: Phase 1
% Craft 2: Phase 4
% Craft 3: Phase 7
% Craft 4: Phase 10

for n = 1:4

  iphase = 3*(n-1)+1;

  % define parameters and states
  p  = input.auxdata.p0;
P1 = input.auxdata.P10;
P2 = input.auxdata.P20;
L  = input.phase(iphase).state(:,1);

  % dynamics of the coast phase
  Ldot = sqrt(mu*p)./p.^2.*(1+P1.*sin(L)+P2.*cos(L)).^2;

  output(iphase).dynamics  = Ldot;

end

%%% Burn Phase
% The middle phase is the burn phase for each craft. There are no parameters, seven states \((p, P_1, P_2, Q_1, Q_2, L, \text{ and } m)\), and three controls \((u_r, u_t, \text{ and } u_n)\).

% Craft 1: Phase 2
% Craft 2: Phase 5
% Craft 3: Phase 8
% Craft 4: Phase 11

for \(n = 1:4\)

    iphase = 3*(n-1)+2;

    % define states and controls (time included for size reference for mdot - see below)
    \(t = \text{input.phase}(iphase).time;\)
    \(p = \text{input.phase}(iphase).state(:,1);\)
    \(P1 = \text{input.phase}(iphase).state(:,2);\)
    \(P2 = \text{input.phase}(iphase).state(:,3);\)
    \(Q1 = \text{input.phase}(iphase).state(:,4);\)
    \(Q2 = \text{input.phase}(iphase).state(:,5);\)
    \(L = \text{input.phase}(iphase).state(:,6);\)
    \(m = \text{input.phase}(iphase).state(:,7);\)
    \(u_r = \text{input.phase}(iphase).control(:,1);\)
    \(u_t = \text{input.phase}(iphase).control(:,2);\)
    \(u_n = \text{input.phase}(iphase).control(:,3);\)

    % path definition
    \(\text{path} = u_r.^2 + u_t.^2 + u_n.^2;\)

    % dynamics of the burn phase
    \(w = 1+P1.*\sin(L)+P2.*\cos(L);\)
    \(sSquare = 1+Q1.^2+Q2.^2;\)
    \(ar = T./m.*u_r;\)
    \(at = T./m.*u_t;\)
    \(an = T./m.*u_n;\)
    \(pdot\) \(= 2*p./w.*\text{sqrt}(p./\mu).*at;\)
    \(P1dot\) \(= \text{sqrt}(p./\mu).*(-ar.*\cos(L)+((w+1).*\sin(L)+P1).*at./w...\)
    \(\quad + (Q2.*\sin(L)-Q1.*\cos(L)).*P2.*an./w);\)
    \(P2dot\) \(= \text{sqrt}(p./\mu).*((ar.*\sin(L)+((w+1).*\cos(L)+P2).*at./w...\)
    \(\quad - (Q2.*\sin(L)-Q1.*\cos(L)).*P1.*an./w);\)
    \(Q1dot\) \(= \text{sqrt}(p./\mu).*sSquare./(2*w).*an.*\sin(L);\)
    \(Q2dot\) \(= \text{sqrt}(p./\mu).*sSquare./(2*w).*an.*\cos(L);\)
    \(LdotT\) \(= \text{sqrt}(\mu.*p).*(w./p).^2+\text{sqrt}(p./\mu).*Q2.*\sin(L)...\)
    \(\quad - Q1.*\cos(L)).*an./w;\)
    \(\text{mdot}\) \(= -T/(g0*Isp)*\text{ones(size(t))};\)

    output(iphase).dynamics = [pdot,P1dot,P2dot,Q1dot,Q2dot,LdotT,mdot];
    output(iphase).path = path;

end

%% Coast Phase
% The first and last phases are the coast phases for each craft. There are
% three parameters (p, P1, and P2) and one state (L). There are no
% controls.

% Craft 1: Phase 3, parameters 1-3
% Craft 2: Phase 6, parameters 4-6
% Craft 3: Phase 9, parameters 7-9
% Craft 4: Phase 12, parameters 10-12

for n = 1:4

    iphase = 3*(n-1)+3;

    p  = input.phase(iphase).parameter(:,3*(n-1)+1);
    P1 = input.phase(iphase).parameter(:,3*(n-1)+2);
    P2 = input.phase(iphase).parameter(:,3*(n-1)+3);
    L  = input.phase(iphase).state(:,1);

    % dynamics of the coarse phase
    Ldot = sqrt(mu*p)./p.^2.*(1+P1.*sin(L)+P2.*cos(L)).^2;

    output(iphase).dynamics = Ldot;

end

end
APPENDIX G

The plotting functions used to generate the figures shown in the Results sections are given below.

```matlab
%% plotOneCraft.m

% Plot one spacecraft results given scaled data in MEE.

function plotOneCraft(output, auxdata, LBar)
    close all
    
    % Assemble data
    phaseCount = 3;
    
    % Preallocate structures
    state = cell(1, phaseCount);
    time = cell(1, phaseCount);
    control = cell(1, phaseCount);
    p = cell(1, phaseCount);
    P1 = cell(1, phaseCount);
    P2 = cell(1, phaseCount);
    Q1 = cell(1, phaseCount);
    Q2 = cell(1, phaseCount);
    L = cell(1, phaseCount);
    m = cell(1, phaseCount);
    r = cell(1, phaseCount);
    rx = cell(1, phaseCount);
    ry = cell(1, phaseCount);
    rz = cell(1, phaseCount);
    v = cell(1, phaseCount);
    
    for iphase = 1:phaseCount
        state{iphase} = output.result.solution.phase(iphase).state;
        time{iphase} = output.result.solution.phase(iphase).time;
        control{iphase} = output.result.solution.phase(iphase).control;
    end
    
    % Phase 1 = [L]
    % Phase 2 = [p P1 P2 Q1 Q2 L m]
    % Phase 3 = [L]
    
    % Phase 1
    iphase = 1;
    p{iphase} = auxdata.p0*ones(size(time{iphase}));
    P1{iphase} = auxdata.P10*ones(size(time{iphase}));
    P2{iphase} = auxdata.P20*ones(size(time{iphase}));
    Q1{iphase} = auxdata.Q10*ones(size(time{iphase}));
    Q2{iphase} = auxdata.Q20*ones(size(time{iphase}));
    L{iphase} = state{iphase}(:,1);
```
m{iphase} = auxdata.SCALEDm0*ones(size(time{iphase}));

% Phase 2
iphase = 2;
p{iphase} = state{iphase}(:,1);
P1{iphase} = state{iphase}(:,2);
P2{iphase} = state{iphase}(:,3);
Q1{iphase} = state{iphase}(:,4);
Q2{iphase} = state{iphase}(:,5);
L{iphase} = state{iphase}(:,6);
m{iphase} = state{iphase}(:,7);

% Phase 3
iphase = 3;
p{iphase} = output.result.solution.parameter(1)*ones(size(time{iphase}));
P1{iphase} = output.result.solution.parameter(2)*ones(size(time{iphase}));
P2{iphase} = output.result.solution.parameter(3)*ones(size(time{iphase}));
Q1{iphase} = Q1{iphase-1}(end)*ones(size(time{iphase}));
Q2{iphase} = Q2{iphase-1}(end)*ones(size(time{iphase}));
L{iphase} = state{iphase}(:,1);
m{iphase} = m{iphase-1}(end)*ones(size(time{iphase}));

% Convert SCALED EE to RV and Unscale
for iphase = 1:phaseCount
    [rr,vv] = mee2rv_old(p{iphase},P1{iphase},P2{iphase},Q1{iphase},Q2{iphase},L{iphase},auxdata.SCALEDmu);
    r{iphase} = rr; % rr = [rx ry rz]
    rx{iphase} = r{iphase}{1}*LBar/1000;
    ry{iphase} = r{iphase}{2}*LBar/1000;
    rz{iphase} = r{iphase}{3}*LBar/1000;
    v{iphase} = vv;
end

% Make Plots

% 3D Plot of Position
figure(1)
plot3(rx{1},ry{1},rz{1},'b',rx{2},ry{2},rz{2},'r.-',rx{3},ry{3},rz{3},'k:','MarkerSize',5)
axis equal
xlabel('Rx (km)')
ylabel('Ry (km)')
zlabel('Rz (km)')
legend('Phase 1 (Coast)','Phase 2 (Burn)','Phase 3 (Coast)')
end
% Want to plot rx, ry, and rz given data in MEE

function plotFourCraft(output,auxdata)

%% Assemble data
phaseCount = 12;

% Preallocate structures
state   = cell(1,phaseCount);
time    = cell(1,phaseCount);
p       = cell(1,phaseCount);
P1      = cell(1,phaseCount);
P2      = cell(1,phaseCount);
Q1      = cell(1,phaseCount);
Q2      = cell(1,phaseCount);
L       = cell(1,phaseCount);
m       = cell(1,phaseCount);
r       = cell(1,phaseCount);
rx      = cell(1,phaseCount);
ry      = cell(1,phaseCount);
rz      = cell(1,phaseCount);
v       = cell(1,phaseCount);
% vx      = cell(1,phaseCount);
% vy      = cell(1,phaseCount);
% vz      = cell(1,phaseCount);

for iphase = 1:phaseCount
    state{iphase}   = output.result.solution.phase(iphase).state;
    time{iphase}    = output.result.solution.phase(iphase).time;
end

% Phase 1 = [L]
% Phase 2 = [p P1 P2 Q1 Q2 L m]
% Phase 3 = [L]

% Phase 1
for iphase = [1 4 7 10]
    p{iphase}  = auxdata.p0*ones(size(time{iphase}));
    P1{iphase} = auxdata.P10*ones(size(time{iphase}));
    P2{iphase} = auxdata.P20*ones(size(time{iphase}));
    Q1{iphase} = auxdata.Q10*ones(size(time{iphase}));
    Q2{iphase} = auxdata.Q20*ones(size(time{iphase}));
    L{iphase}  = state{iphase}(:,1);
    m{iphase}  = auxdata.SCALEDm0*ones(size(time{iphase}));
end

% Phase 2
for iphase = [2 5 8 11]
    p{iphase}  = state{iphase}(:,1);
    P1{iphase} = state{iphase}(:,2);
    P2{iphase} = state{iphase}(:,3);
    Q1{iphase} = state{iphase}(:,4);
    Q2{iphase} = state{iphase}(:,5);
    L{iphase}  = state{iphase}(:,6);
end

57
m{iphase} = state{iphase}(:,7);
end

% Phase 3
for iphase = [3 6 9 12]
  p{iphase} = p{iphase-1}(end)*ones(size(time{iphase}));
P1{iphase} = P1{iphase-1}(end)*ones(size(time{iphase}));
P2{iphase} = P2{iphase-1}(end)*ones(size(time{iphase}));
Q1{iphase} = Q1{iphase-1}(end)*ones(size(time{iphase}));
Q2{iphase} = Q2{iphase-1}(end)*ones(size(time{iphase}));
L{iphase} = state{iphase}(:,1);
m{iphase} = m{iphase-1}(end)*ones(size(time{iphase}));
end

%% Convert EE to RV and plot
% Convert EE to RV on phase basis
for iphase = 1:phaseCount
  [rr,vv] = ee2rv(p{iphase},P1{iphase},P2{iphase},Q1{iphase},Q2{iphase},L{iphase},auxdata.SCALEDmu);
  r{iphase} = rr;
  rx{iphase} = r{iphase}{1};
  ry{iphase} = r{iphase}{2};
  rz{iphase} = r{iphase}{3};
  v{iphase} = vv;
end

% 3-axis plots
figure
subplot(2,2,1)
plot3(rx{1},ry{1},rz{1},'b', rx{2},ry{2},rz{2},'r', rx{3},ry{3},rz{3},'g')
axis equal
xlabel('rx');
ylabel('ry');
zlabel('rz');
title('Craft 1')

subplot(2,2,2)
plot3(rx{4},ry{4},rz{4},'b', rx{5},ry{5},rz{5},'r', rx{6},ry{6},rz{6},'g')
axis equal
xlabel('rx');
ylabel('ry');
zlabel('rz');
title('Craft 2')

subplot(2,2,3)
plot3(rx{7},ry{7},rz{7},'b', rx{8},ry{8},rz{8},'r', rx{9},ry{9},rz{9},'g')
axis equal
xlabel('rx');
ylabel('ry');
zlabel('rz');
title('Craft 3')
subplot(2,2,4)
plot3(rx{10},ry{10},rz{10},'b', rx{11},ry{11},rz{11},'r',
rx{12},ry{12},rz{12},'g')
axis equal
xlabel('rx');
ylabel('ry');
zlabel('rz');
title('Craft 4')
end

function plotEnd(output,auxdata,LBar)

  %% Assemble Data
  phaseCount = l2;

  % preallocate structures
  state    = cell(1,phaseCount);
  time     = cell(1,phaseCount);
  p        = cell(1,phaseCount);
  P1       = cell(1,phaseCount);
  P2       = cell(1,phaseCount);
  Q1       = cell(1,phaseCount);
  Q2       = cell(1,phaseCount);
  L        = cell(1,phaseCount);
  m        = cell(1,phaseCount);
  r        = cell(1,phaseCount);
  rx       = cell(1,phaseCount);
  ry       = cell(1,phaseCount);
  rz       = cell(1,phaseCount);
  v        = cell(1,phaseCount);

  for iphase = 1:phaseCount
    state{iphas} = output.result.solution.phase(iphas).state;
    time{iphas}  = output.result.solution.phase(iphas).time;
  end

  % Phase 1 = [L]
  % Phase 2 = [p P1 P2 Q1 Q2 L m]
  % Phase 3 = [L]

  % Phase 1
  for iphase = [1 4 7 10]
    p{iphas} = auxdata.p0*ones(size(time{iphas}));
    P1{iphas} = auxdata.P10*ones(size(time{iphas}));
    P2{iphas} = auxdata.P20*ones(size(time{iphas}));
    Q1{iphas} = auxdata.Q10*ones(size(time{iphas}));
    Q2{iphas} = auxdata.Q20*ones(size(time{iphas}));
    L{iphas}  = state{iphas}(:,1);
  end
m{iphase} = auxdata.SCALEDm0*ones(size(time{iphase}));
end

% Phase 2
for iphase = [2 5 8 11]
p{iphase} = state{iphase}(:,1);
P1{iphase} = state{iphase}(:,2);
P2{iphase} = state{iphase}(:,3);
Q1{iphase} = state{iphase}(:,4);
Q2{iphase} = state{iphase}(:,5);
L{iphase} = state{iphase}(:,6);
m{iphase} = state{iphase}(:,7);
end

% Phase 3
for iphase = [3 6 9 12]
p{iphase} = p{iphase-1}(end)*ones(size(time{iphase}));
P1{iphase} = P1{iphase-1}(end)*ones(size(time{iphase}));
P2{iphase} = P2{iphase-1}(end)*ones(size(time{iphase}));
Q1{iphase} = Q1{iphase-1}(end)*ones(size(time{iphase}));
Q2{iphase} = Q2{iphase-1}(end)*ones(size(time{iphase}));
L{iphase} = state{iphase}(:,1);
m{iphase} = m{iphase-1}(end)*ones(size(time{iphase}));
end

%% Convert EE to RV on phase basis and unscale
for iphase = 1:phaseCount
[rr,vv] = ee2rv(p{iphase},P1{iphase},P2{iphase},Q1{iphase},Q2{iphase},L{iphase},auxdata.SCALEDmu);
r{iphase} = rr;
rx{iphase} = r{iphase}{1}*LBar/1000;
ry{iphase} = r{iphase}{2}*LBar/1000;
rz{iphase} = r{iphase}{3}*LBar/1000;
v{iphase} = vv;
end

%% Plot Final Position on One Graph
figure
plot3(rx{3}(end),ry{3}(end),rz{3}(end),'bo',
rx{6}(end),ry{6}(end),rz{6}(end),'gs',
rx{9}(end),ry{9}(end),rz{9}(end),'rd',rx{12}(end),ry{12}(end),rz{12}(end),'cv ')
axis equal
xlabel('Rx (km)');
ylabel('Ry (km)');
zlabel('Rz (km)');
legend('Spacecraft 1','Spacecraft 2','Spacecraft 3','Spacecraft 4')
% Plot controls for either one or four spacecraft

function plotControls(output,phaseCount,TBar)

%% Assemble Data
craftCount = phaseCount/3;

% Preallocate structures
time = cell(1,phaseCount);
control = cell(1,phaseCount);

for iphase = 1:phaseCount
    time{iphase} = output.result.solution.phase(iphase).time;
    control{iphase} = output.result.solution.phase(iphase).control;
end

controlString = {'ur' 'ut' 'un'};
colors = {'b-' 'g:' 'r-.'};

%% One Craft
if craftCount == 1
    %Plot all controls vs. time
    figure
    unscaledTime = time{2}*TBar;
    plot(unscaledTime,control{2}(:,1),'b-',unscaledTime,control{2}(:,2),'k:',
    unscaledTime,control{2}(:,3),'r-.'),xlabel('Time (s)'),ylabel('Command')
    legend('ur','ut','un','Location','East')

    % Plot each control (ur, ut, and un) vs. time
    figure
    for n = 1:3
        subplot(1,3,n)
        plot(time{2},control{2}(:,n),colors{n}),xlabel('Time (s)'),ylabel('Command Effort')
        title(controlString{n})
    end
end

%% Four Crafts
if craftCount == 4
    titleString = {'Spacecraft 1' 'Spacecraft 2' 'Spacecraft 3' 'Spacecraft 4'};

    %Plot all controls vs. time for each craft
    for n = 1:craftCount

% figure(n)
subplot(2,2,n)
iphase = 3*n-1;
unscaledTime = time{iphase}*TBar;
plot(unscaledTime,control{iphase}(:,1),'b-',unscaledTime,control{iphase}(:,2),'k:',unscaledTime,control{iphase}(:,3),'r-.'
xlabel('Time (s)')
ylabel('Command')
legend(['ur','ut','un','Location','East'])
title(titleString{n})
end
end
end