THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF AEROSPACE ENGINEERING

AN EVOLUTIONARY ALGORITHM FOR LOW-THRUST ORBITAL TRANSFERS

KAOMI HASHIMOTO
Spring 2012

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Aerospace Engineering
with honors in Aerospace Engineering

Reviewed and approved* by the following:

Dr. Robert Melton
Professor of Aerospace Engineering
Thesis Supervisor

Dr. George Lesieutre
Professor and Head,
Aerospace Engineering
Honors Adviser

Dr. David Spencer
Associate Professor of Aerospace Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.
Abstract

Flocks of birds seem to travel randomly when searching for food, but each bird is actually communicating its route with the others in order to find the optimal path to their location. The particle swarm optimization (PSO) technique is a stochastic method that utilizes this theory to optimize an unknown parameter by analyzing the behavior of a swarm of particles. In this thesis, the PSO technique has been applied to create an evolutionary algorithm that analyzes low-thrust transfers between two co-planar, elliptical orbits, both coaxial and non-coaxial. Transfers are modeled as two thrust arcs with an intermediary coast arc. The algorithm determines the optimal thrust pointing angles, the initial true anomaly of the spacecraft, the duration of each thrust arc, and the change in eccentric anomaly of the coast arc that result in the greatest final-to-initial mass ratio. While the method is successful in producing the optimal position vector for the transfer, it is imperative that the penalties and penalty conditions placed on the objective function constraints are also optimized. The PSO algorithm is successful in determining a transfer trajectory between non-coaxial elliptical orbits, but the accuracy of these results is still unknown.
Table of Contents

LIST OF FIGURES ................................................................................................................ III
LIST OF TABLES .................................................................................................................... IV
ACKNOWLEDGEMENTS ........................................................................................................ V

1. INTRODUCTION ................................................................................................................... 1
   1.1 REVIEW OF LITERATURE ............................................................................................. 1
   1.2 TRAJECTORY OPTIMIZATION .................................................................................... 2

2. PROBLEM STATEMENT ....................................................................................................... 3

3. METHODOLOGY .................................................................................................................. 7

4. RESULTS AND DISCUSSION ............................................................................................. 10

5. CONCLUSIONS AND FUTURE WORK ............................................................................. 17

REFERENCES ....................................................................................................................... 18

APPENDIX A: ELLIPSE MATLAB CODE ............................................................................. 19
   A.1 MAIN CODE .................................................................................................................. 19
      A.1.1 PSO_Ellipses.m ....................................................................................................... 19
   A.2 FUNCTIONS .................................................................................................................. 20
      A.2.1 EvalJ_Ellipse.m ..................................................................................................... 20
      A.2.2 EvalPGBest_Ellipse.m .......................................................................................... 23
      A.2.3 PSOode.m .............................................................................................................. 23
      A.2.4 PSOode2.m ........................................................................................................... 24
      A.2.5 UpdateP_Ellipse.m .............................................................................................. 24
      A.2.6 UpdateV_Ellipse.m .............................................................................................. 24

APPENDIX B: BASE MATLAB CODE .................................................................................. 26
   B.1 MAIN CODE .................................................................................................................. 26
      B.1.2 PSO_base.m .......................................................................................................... 26
   B.2 FUNCTIONS .................................................................................................................. 27
      B.2.1 EvalJ_base.m ......................................................................................................... 27
      B.2.2 EvalPGBest_base.m ............................................................................................... 29
      B.2.3 PSOode.m .............................................................................................................. 29
      B.2.4 PSOode2.m ............................................................................................................ 30
      B.2.5 UpdateP_base.m ................................................................................................... 30
      B.2.6 UpdateV_base.m .................................................................................................. 31

APPENDIX C: NON-COAXIAL MATLAB CODE .................................................................. 32
   C.1 EvalJ_NONAX.m .......................................................................................................... 32
List of Figures

FIGURE 1. NON-COAXIAL ELLIPSES................................................................. 9
FIGURE 2. NON-COAXIAL RESULTS FOR $\psi = \pi/2$ ........................................... 14
FIGURE 3. MULTIPLE TRIALS FOR $\psi = \pi/2$ AND $E = 0.1$ .................................... 15
List of Tables

TABLE 1. FINAL-TO-INITIAL MASS RATIO .................................................................................................. 11
TABLE 2. PERCENT ERROR OF PSO SOLUTIONS ............................................................................. 11
TABLE 3. RESULTS OF DIFFERENT PENALTY/CONDITION COMBINATIONS ............................. 12
TABLE 4. NON-COAXIAL RESULTS FOR $\Psi = \Pi/2$ ............................................................................... 13
TABLE 5. MULTIPLE TRIALS FOR $\Psi = \Pi/2$ AND $E = 0.1$ ................................................................. 15
Acknowledgements

I would first like to thank Dr. Robert Melton for his continued support over the past four years. As both my professor and my thesis supervisor, he has greatly expanded my knowledge of the field of aerospace engineering. His wisdom and willingness to work closely with students has been a source of great respect and admiration. Secondly, I would like to thank Dr. George Lesieutre and Dr. David Spencer for taking the time to read my thesis; it is greatly appreciated. Finally, I would like to thank the entire Department of Aerospace Engineering for providing me with my education and a vast network of resources that will be highly valuable in my future.
1. Introduction

The particle swarm optimization (PSO) technique is an evolutionary algorithm based upon the patterns followed by flocks of birds when searching for food. As a flock travels, it seems as if there is no specific flying pattern or that there is not a single bird leading the flock. Through further investigation, however, it is noticed that each bird is adapting its path based upon its best position and the flock’s best position relative to the food source.

The PSO algorithm is a population-based stochastic method\cite{1} that utilizes swarm behavior in order to optimize a particular solution. The problem must calculate a number of parameters that minimize an objective function, whose expression is composed of expected orbital properties and penalties for incorrect value. Each particle within the swarm has a position in $N$-dimensional space that is defined by $N$ parameters that are initially randomly populated. The random values are normally distributed between the upper and lower limits of the variables. These limits are used to create the limits of the particle’s velocity, which updates its position between iterations. The algorithm stores each particle’s best position and the swarm’s best position in order to update the position vectors for the following iteration\cite{5}.

The unknown parameters in the position vector are substituted into the objective function, usually called $J$. For each application of the PSO algorithm, specified constraints are used to calculate $J$. There is one $J$-value for every particle within a swarm, and the algorithm finds the optimal position vector by minimizing $J$. This is done through multiple iterations of evolving swarms until $J$ converges to its minimum value.

1.1 Review of Literature

The particle swarm optimization technique is highly versatile and has been applied to many different aerospace problems in all three divisions of aerospace engineering. It has been used to design optimal helicopter rotor blades by maintaining a specified stiffness value and maximizing elastic coupling\cite{4}. Additionally, the PSO algorithm can be used in the design of small satellite launch vehicles. The algorithm constrains the objective function using characteristics of propulsion, structure, and aerodynamics\cite{1}. In addition to the orbital mechanics application of the algorithm used in this thesis, the PSO method can also be applied to dynamic attitude...
This application of the particle swarm method is based upon the work of Pontani and Conway\textsuperscript{[2]}. They use the PSO algorithm to calculate the optimal finite-thrust transfer between coplanar, circular orbits, i.e. a Hohmann transfer with finite thrusts instead of impulsive thrusts. Since a Hohmann transfer is known to be the most propellant-efficient transfer between circular orbits, the results from the code could be compared and verified. Although evolutionary algorithms are better suited to the optimization of impulsive trajectories\textsuperscript{[3]}, it is found that the PSO algorithm returns an accurate position vector for minimizing propellant consumption.

1.2 Trajectory Optimization

A major parameter in spacecraft design is minimizing mass. Spacecraft, however, must often transfer between multiple orbits in order to achieve their final desired orbit. To do this, they must fire their thrusters and expel propellant. Since propellant equals additional required mass, the optimal transfer trajectory between orbits is the one that minimizes the required $\Delta V$, and thus minimizes the necessary propellant.

This thesis analyzes the problem of transferring a spacecraft between two coplanar, elliptical orbits. The algorithm has been applied to both ellipses that share a line of apsides and non-coaxial ellipses. The orbital transfer is assumed to be a finite thrust arc out of an inner orbit, followed by a Keplerian coast arc, and ending with a second finite thrust arc to enter the larger (in regards to semi-major axis) orbit. The PSO algorithm, coded in MATLAB, determines the length of time of each thrust arc, the direction of the thrust, and the change in eccentric anomaly that occurs during the coast arc. The optimal position vector will minimize the propellant consumption by maximizing the final-to-initial mass ratio\textsuperscript{[2]}. 


2. Problem Statement

Pontani and Conway apply the particle swarm optimization theory to the finite-thrust orbital transfer between two circular orbits. The goal is to create an algorithm that determines the most favorable trajectory between orbits by optimizing the final-to-initial mass ratio. These results are compared to those calculated by a Hohmann transfer, the known solution for optimizing the propellant necessary for transfers between circular orbits.

The Hohmann-like transfer is modeled as a thrust arc out of the first orbit for a specified length of time, a coast arc for a specified change in eccentric anomaly, and a second thrust arc into the second orbit for a specified time interval. A particle’s position is composed of eleven parameters: eight coefficients ($\zeta_0, \zeta_1, \zeta_2, \zeta_3, \theta_0, \theta_1, \theta_2, \theta_3$) used to determine the thrust pointing angle for the calculation of $\dot{v}_r$ and $\dot{v}_q$, the time length of each thrust arc ($\Delta t_1$ and $\Delta t_2$), and the change in eccentric anomaly, $\Delta E$, of the coast arc. The position vector is defined by

$$\mathbf{\chi} = \begin{bmatrix} \zeta_0 & \zeta_1 & \zeta_2 & \zeta_3 & \theta_0 & \theta_1 & \theta_2 & \theta_3 & \Delta t_1 & \Delta E & \Delta t_2 \end{bmatrix}^T$$

(1)

Since the two orbits are circular, the constraints for the objective function take into account the fact that a spacecraft’s velocity on a circular orbit is only in the tangential direction.

In this thesis, however, the PSO algorithm is manipulated for elliptical orbits, thus changing the constraints for the objective function. The eccentricities of the orbits are equal and pre-defined, and the semi-major axes of the two orbits are related by the ratio $\beta = a_2/a_1$. An additional term, $\theta_0$, is added to the position vector to represent where along the inner orbit the spacecraft first starts thrusting:

$$\mathbf{\chi} = \begin{bmatrix} \zeta_0 & \zeta_1 & \zeta_2 & \zeta_3 & \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_0 & \Delta t_1 & \Delta E & \Delta t_2 \end{bmatrix}^T$$

(2)

Using the eccentricity and semi-major axis of the inner orbit, the semi-latus rectum, radius, velocity, and velocity components of the spacecraft at $t = 0$ can be found using the following equations (the “0” subscript represents the spacecraft’s properties at $t = 0$, and the “1” subscript represents the properties of the inner orbit):
\[ p_1 = a_1(1 - e_1^2) \]  

(3)

\[ n_0 = \frac{p_1}{1 + e_1 \cos \theta_0} \]  

(4)

\[ v_0 = \sqrt{2 \left( 1 + \frac{B}{r_0} \right)} \]  

(5)

\[ v_{r0} = \frac{B}{\sqrt{a_1(1 - e_1^2)}} e_1 \sin \theta_0 \]  

(6)

\[ v_\theta = \frac{B}{\sqrt{a_1(1 - e_1^2)}} (1 + e_1 \cos \theta_0) \]  

(7)

\[ \dot{\theta}_0 = \frac{v_\theta}{r_0} \]  

(8)

The trajectories of the thrust arcs are determined by considering the thrust-to-mass ratios \( (T/m) \) of the spacecraft and the thrust pointing angles, which are based upon the eight unknown coefficients of the position vector. In order to calculate the thrust-to-mass ratio, two assumptions must be made\(^{[2]}\):

(a) maximum thrust is employed during the two thrust arcs

(b) in each thrust arc, the thrust pointing angle is represented with a third degree polynomial function of time:

\[
\delta = \xi_0 + \xi_1 t + \xi_2 t^2 + \xi_3 t^3 \quad \text{if} \quad 0 \leq t \leq t_1
\]

\[
\delta = \vartheta_0 + \vartheta_1(t-t_2) + \vartheta_2(t-t_2)^2 + \vartheta_3(t-t_2)^3 \quad \text{if} \quad t_1 \leq t \leq t_2
\]

(9)

These assumptions lead to the thrust-to-mass ratio being defined as

\[
\frac{T}{m} = \begin{cases} 
\frac{T}{m_0} = \frac{c n_0 t}{c - n_0 t} & \text{if} \quad 0 \leq t \leq t_1 \\
\frac{T}{m_0} = \frac{c n_0}{t - n_0(t_1 + t_2)} & \text{if} \quad t_1 \leq t \leq t_2
\end{cases}
\]

(10)

where \( T \) symbolizes the thrust level; \( c \) is the effective exhaust velocity; and \( n_0 \) is the initial thrust-to-mass ratio. \( t_1 \) represents the time at which the first thrust arc terminates (i.e. the coast arc
begins); \( t_2 \) is the time that the second thrust arc begins (or the end of the coast arc); and \( t_f \) is the end of the orbital transfer.

In order to model the spacecraft’s motion, two ODE’s are required to relate the thrust arcs to time and the thrust-pointing angle. The equations of motion of the spacecraft have the following expressions:

\[
\begin{align*}
\dot{r} &= v_r \\
\dot{v}_r &= \frac{B + rv^2}{r^2} + \frac{T}{m} \sin \theta \\
\dot{\theta} &= \frac{v}{r} \\
\dot{v} &= \frac{2v_r v}{r} + \frac{T}{m} \cos \theta
\end{align*}
\]

where the state variables are given by

\[
\begin{align*}
\dot{x}_1 &= \dot{r} \\
\dot{x}_2 &= \dot{v}_r \\
\dot{x}_3 &= \dot{\theta} \\
\dot{x}_4 &= \dot{v}
\end{align*}
\]

The coast arc between the thrust arcs forms part of an elliptical Keplerian orbit. The orbital properties of the coast arc (denoted with the subscript “coast”) can be found using the radius and velocity components at time \( t_1 \):

\[
a_{\text{coast}} = \frac{\frac{B \eta}{\eta(v^2 + v_1^2)}}{2} \\
e_{\text{coast}} = \sqrt{1 - \frac{\eta^2 v_1^2}{B a_{\text{coast}}}}
\]

As the change in eccentric anomaly of the coast arc is defined by the PSO, the eccentric anomaly after the first thrust arc must be found using the spacecraft’s true anomaly at time \( t = t_1 \) using the tangential relationship between eccentric and true anomaly.

\[
\tan \frac{1}{2} = \sqrt{\frac{1 + e_{\text{coast}}}{1 - e_{\text{coast}}}} \tan \frac{E_1}{2}
\]

\( t_2 \) signifies the end of the coast arc and the beginning of the second thrust arc. Since \( \Delta E \) is a parameter in a particle’s position vector, the spacecraft’s eccentric anomaly at the end of the coast arc is given by \( E_2 = E_1 + \Delta E \) and can be converted back to true anomaly using Eq. (15). The
variables in Eqs. (3-8) can be adjusted to reflect the properties of the coast arc and the spacecraft’s position at $t_2$:

$$p_{coast} = a_{coast} \left( 1 - e_{coast}^2 \right)$$  \hspace{1cm} (16)

$$r_2 = \frac{p_{coast}}{1 + e_{coast} \cos \frac{q_2}{2}}$$  \hspace{1cm} (17)

$$v_{r_2} = \sqrt{\frac{B}{a_{coast} \left( 1 - e_{coast}^2 \right)}} e_{coast} \sin \frac{q_2}{2}$$  \hspace{1cm} (18)

$$v_{\frac{2}{2}} = \sqrt{\frac{B}{a_{coast} \left( 1 - e_{coast}^2 \right)}} \left( 1 + e_{coast} \cos \frac{q_2}{2} \right)$$  \hspace{1cm} (19)

$$\dot{q}_2 = \frac{v_{\frac{2}{2}}}{r_2}$$  \hspace{1cm} (20)

This application of the PSO algorithm aims to maximize the final-to-initial mass ratio ($m_f/m_0$) of the spacecraft by minimizing the objective function ($J$). The expression for the objective function is dependent upon the constraints created by the problem. These equations are given by

$$\frac{m_f}{m_0} = \frac{m_0}{c} \left( \frac{T}{c} - \Delta t_1 - \Delta t_2 \right) = 1 - \frac{n_0}{c} \left( \Delta t_1 + \Delta t_2 \right)$$  \hspace{1cm} (21)

$$J = \Delta t_1 + \Delta t_2$$  \hspace{1cm} (22)
3. Methodology

The code developed for the elliptical orbit problem utilizes six functions. The entire PSO code contains the following (which can be found in the appendices):

- A main code (*PSO_Ellipses.m*)
- A function that completes the calculations necessary in order to optimize the problem, including the calculation of the objective function (*EvalJ_Ellipse.m*)
- Two functions (one for each thrust arc) that determine the numerical solutions to the state variable differential equations (*PSOode.m* and *PSOode2.m*)
- A function that examines the objective function values (*J*-values) of the swarm (*EvalPGBest_Ellipse.m*)
- A function that updates a particle’s velocity vector (*UpdateV_Ellipse.m*)
- A function that updates a particle’s position vector (*UpdateP_Ellipse.m*)

The random initialization of the swarm occurs in the main code. The position vectors of the swarm follow a normal distribution within the constraints of each parameter. The following items are also declared in the main code: the number of particles in a swarm, the number of parameters in the position vector, the number of iterations that will occur, and the upper and lower limits of both the position and velocity vectors. As each iteration is completed, the swarm evolves and adapts to minimize *J* and find the optimal solution. For this problem, a swarm is defined as 50 particles, and 2000 iterations are completed. (1500 iterations are completed for the non-coaxial problem.) The parameters have the following upper and lower limits:

\[-\pi/2 \leq \theta_0 \leq \pi/2 \quad 0 \text{ TU} \leq \Delta t_1 \leq 3 \text{ TU} \quad 0 \leq \Delta E \leq 2\pi \quad 0 \text{ TU} \leq \Delta t_2 \leq 3 \text{ TU} \]

\[-1 \leq \zeta_k \leq 1 \quad -1 \leq \vartheta_k \leq 1 \quad (k=0,1,2,3)\]  \hspace{1cm} (23)

The optimal value for the parameters falls in these ranges because the code employs canonical units for all its variables. One distance unit (DU) is the length of the semi-major axis of the inner orbit, and a time unit (TU) is the value that creates \( \mu_B = 1 \text{ DU}^3/\text{TU}^2 \). For this problem, the effective exhaust velocity, \( c \), is set to 0.5 DU/TU, and the initial thrust-to-mass ratio, \( n_0 \), is 0.2 DU/TU^2.
There are certain constraints that must be met in order to ensure the optimality of the solution. For the coaxial problem, there are three conditions:

(i) The two elliptical orbits must share a common line of apsides (LOA)

(ii) The semi-major axis of the outer orbit must relate to that of the inner orbit by the ratio $\beta$

(iii) The final orbit’s eccentricity must match the pre-defined final eccentricity

If a particle’s position vector does not meet these conditions to within a specified tolerance, there are penalties applied to the objective function, which is now expressed as:

$$J = \Delta t_1 + \Delta t_2 + \sum_{k=1}^{3} \alpha_k |d_k|$$  \hspace{1cm} (23)

where $\alpha$ represents the penalty applied to each constraint, and $d$ is the position vector’s error corresponding to each condition. Based on conditions (i – iii), $d_k$ is defined by the following expressions:

$$d_1 = f - q_{\text{new}}$$  
$$d_2 = a_{2 \text{calc}} - a_2$$  
$$d_3 = e_{2 \text{calc}} - e_2$$  \hspace{1cm} (24)

where $\phi$ is the true anomaly at $t = t_f$ with respect to the inner orbit’s LOA, $\theta_{\text{new}}$ is the true anomaly at $t = t_f$ with respect to the outer orbit’s LOA, $a_{2 \text{calc}}$ is the calculated semi-major axis of the final outer orbit, and $e_{2 \text{calc}}$ is the calculated eccentricity of the final outer orbit. The penalties and tolerances associated with each constraint are originally defined as:

$$a_1 = 150 \text{ if } |d_1| < 10^{-3}$$  
$$a_2 = 200 \text{ if } |d_2| < 10^{-3}$$  
$$a_3 = 200 \text{ if } |d_3| < 10^{-3}$$  \hspace{1cm} (25)

Condition (i) must be changed for the problem where the two orbits are non-coaxial. Constraint $d_1$ is altered to become:

$$d_1 = f - q_{\text{new}} - \Psi$$  \hspace{1cm} (26)

where $\phi$ is the true anomaly at $t = t_f$ with respect to the inner orbit’s LOA, $\theta_{\text{new}}$ is the true anomaly at $t = t_f$ with respect to the outer orbit’s LOA, and $\Psi$ is the angle between the orbits’ lines of apsides, as shown in Figure 1. The penalty conditions are set to:
\begin{align*}
1 &= 100 \quad \text{if } |d_1| < 10^{-3} \\
2 &= 25 \quad \text{if } |d_2| < 10^{-3} \\
1 &= 25 \quad \text{if } |d_3| < 10^{-3}
\end{align*}

\textbf{Figure 1. Non-coaxial Ellipses}

The PSO algorithm stores each particle’s \textit{local} best position vector (correlating to the lowest \(J\)-value) as well as the swarm’s \textit{global} best position vector. The absolute minimum \(J\)-value corresponds to the maximum final-to-initial mass ratio, and therefore, to the problem’s optimal position vector. It is possible, however, for the code to find a local minimum instead of an absolute one. In order to prevent this from happening, the \(J\)-values are analyzed every 100 iterations. If the average \(J\)-value of the previous 10 iterations is less than 1\% smaller or larger than the current global best \(J\)-value (i.e. the \(J\)-value has stagnated), half of the swarm is re-randomized. The code will either re-converge to the same value, or it will find a new minimum \(J\)-value.
4. Results and Discussion

The code created for the finite-thrust transfer between elliptical orbits will be referred to as the ellipse code (found in Appendix A). The code created specifically for the Pontani and Conway Hohmann transfer problem will be referred to as the base code (located in Appendix B).

In order to ensure the ellipse code returns accurate values, it is applied to the Hohmann transfer problem. The final-to-initial mass ratio is compared to that given by the base code and the ideal Hohmann ratio, which is expressed as:

\[
\left(\frac{m_f}{m_0}\right)_H = \exp\left[-\frac{\Delta v_1 + \Delta v_2}{c}\right]
\]  

Because Hohmann transfers refer to circular orbits and circular orbits have zero eccentricities, the constraints and penalties for the objective function are adjusted. The constraints are re-evaluated to be:

\[
d_1 = f-p
\quad d_2 = a_{2,\text{calc}} - a_2
\quad d_3 = e_{2,\text{calc}} - e_2
\quad d_4 = q_0 - 0
\]

Through trial and error, the optimal penalties are found to be:

\[
\begin{align*}
    a_1 &= 150 \quad \text{if} \quad |d_1| < 10^{-5} \\
    a_2 &= 200 \quad \text{if} \quad |d_2| < 10^{-3} \\
    a_3 &= 150 \quad \text{if} \quad |d_3| < 10^{-5} \\
    a_4 &= 150 \quad \text{if} \quad |d_4| < 10^{-6}
\end{align*}
\]

Table 1 summarizes the final-to-initial mass ratios found by each method, and Table 2 shows the percent errors from the known Hohmann solution.
As one can see, the ellipse code returns more accurate results than the base code. This indicates that particle swarm optimization can indeed be applied to finite-thrust transfers between elliptical orbits. This also suggests that an objective function relating the physical parameters of the orbits is preferred to one that monitors the position and velocity of the spacecraft at a given time.

A major factor in determining the optimal solution is the penalties and the conditions under which they are applied. Trial and error was used until the characteristics in Eq. (30) were deemed to return the most accurate results. Several different combinations of conditions and penalties were tested until Eq. (30) was selected. Other combinations that give fairly good results are:

\[ a_1 = 150 \quad \text{if} \quad |d_1| < 10^{-5} \]
\[ a_2 = 150 \quad \text{if} \quad |d_2| < 10^{-3} \]
\[ a_3 = 200 \quad \text{if} \quad |d_3| < 10^{-3} \]
\[ a_4 = 150 \quad \text{if} \quad |d_4| < 10^{-6} \]  

(31)

Table 1. Final-to-Initial Mass Ratio

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>Hohmann Calculation\textsuperscript{[2]}</th>
<th>Base Code</th>
<th>Ellipse Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.566140</td>
<td>0.5024259</td>
<td>0.54494021</td>
</tr>
<tr>
<td>4</td>
<td>0.407642</td>
<td>0.32304846</td>
<td>0.41896981</td>
</tr>
<tr>
<td>6</td>
<td>0.368367</td>
<td>0.27721916</td>
<td>0.40371116</td>
</tr>
<tr>
<td>8</td>
<td>0.353299</td>
<td>0.30621682</td>
<td>0.30903775</td>
</tr>
<tr>
<td>10</td>
<td>0.346603</td>
<td>0.2463908</td>
<td>0.35602986</td>
</tr>
</tbody>
</table>

Table 2. Percent Error of PSO Solutions

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>Base Code</th>
<th>Ellipse Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11.254%</td>
<td>3.745%</td>
</tr>
<tr>
<td>4</td>
<td>20.752%</td>
<td>2.779%</td>
</tr>
<tr>
<td>6</td>
<td>24.744%</td>
<td>9.595%</td>
</tr>
<tr>
<td>8</td>
<td>13.326%</td>
<td>12.528%</td>
</tr>
<tr>
<td>10</td>
<td>28.913%</td>
<td>2.720%</td>
</tr>
</tbody>
</table>
\[ a_i = 150 \quad \text{if} \quad d_i < 10^{-3} \]
\[ a_2 = 150 \quad \text{if} \quad d_2 < 10^{-5} \]
\[ a_3 = 200 \quad \text{if} \quad d_3 < 10^{-5} \]
\[ a_4 = 150 \quad \text{if} \quad d_4 < 10^{-6} \]  

(32)

Although the difference between Eq. (31) and Eq. (32) is small (only the penalty conditions have changed between the two), its effect on the result each obtains is substantial, especially in comparison to Eq. (30). These effects are summarized in Table 3.

**Table 3. Results of Different Penalty/Condition Combinations**

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>Eq. (30) – Values used for final results</th>
<th>Eq. (31)</th>
<th>Eq. (32)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m_f/m_0 )</td>
<td>% Error</td>
<td>( m_f/m_0 )</td>
</tr>
<tr>
<td>2</td>
<td>0.54494021</td>
<td>3.7446%</td>
<td>0.45932925</td>
</tr>
<tr>
<td>4</td>
<td>0.41896981</td>
<td>2.779%</td>
<td>0.13128961</td>
</tr>
<tr>
<td>6</td>
<td>0.40371116</td>
<td>9.595%</td>
<td>0.3856872</td>
</tr>
<tr>
<td>8</td>
<td>0.30903775</td>
<td>12.528%</td>
<td>0.30749861</td>
</tr>
<tr>
<td>10</td>
<td>0.35602986</td>
<td>2.720%</td>
<td>0.31939222</td>
</tr>
</tbody>
</table>

The slight change in penalty conditions between Eq. (31) and Eq. (32) creates a significant rise or fall in percent error. Three of the five \( \beta \)-values see a sharp decrease in error from Eq. (32) to Eq. (31), ranging from 6% to 60%. For the two \( \beta \)-values that increased in error, the shift is only 8% to 21%. With the exception of two \( \beta \)-values, however, Eq. (30) always returns a lower percent error than Eqs. (31-32). Additionally, only two \( \beta \)-values result in an error greater than engineering accuracy. It should be noted that, on average, all 3 penalty and penalty condition combinations used with the ellipse code return a lower percent error than the base code.

These are the differences that resulted from the application of the ellipse code to the Hohmann transfer problem. If the code were applied to orbits with nonzero eccentricities, however, the penalties and conditions, i.e. Eq. (25), would change yet again. Because there are no known analytical solutions for the optimal transfer between elliptical orbits, an iterative algorithm needs to be designed in order to determine the conditions and penalties without using the trial and error method. Furthermore, the results will vary slightly depending upon the operating system of the
computer and the version of MATLAB that is being used.

For non-coaxial elliptical orbits, there are no known analytical solutions for the optimal transfer between orbits. It is expected, however, that more propellant is needed for higher eccentricities (holding the angle between LOA’s constant), resulting in a lower final-to-initial mass ratio. For the non-axial problem, only the EvalJ function is changed, and it can be found in Appendix C. Table 4 summarizes the resulting final-to-initial mass ratios given by the code at varying eccentricities. Figure 2 depicts the trend found by the code.

<table>
<thead>
<tr>
<th>Table 4. Non-coaxial Results for $\Psi = \pi/2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eccentricity</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.3</td>
</tr>
<tr>
<td>0.4</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>0.85</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.95</td>
</tr>
<tr>
<td>0.99</td>
</tr>
</tbody>
</table>
Figure 2. Non-coaxial Results for $\Psi = \pi/2$

Figure 2 follows the expected trend except for the spike in final-to-initial mass ratio when $e > 0.7$. While this is anomalous, the code does return the lowest final-to-initial mass ratio when $e = 0.99$, which is the anticipated result since those orbits are almost parabolic. The deviation from the declination in mass ratio at $e > 0.7$ does not signify that the PSO algorithm is unable to accurately determine the optimal transfer trajectory of non-coaxial ellipses. The results listed in Table 4 were obtained by running the code once, restarting MATLAB every time to ensure the same random particles were generated every time.

Table 5 shows the variation in results when MATLAB is not restarted, and Figure 3 is a graphical representation of the variation. The values used are $\Psi = \pi/2$ and $e_1 = e_2 = 0.1$. 
There is a 60% difference between the values for the first two trials, but the results for Trials 2-8 all lie within 23% of each other. Additionally, the results for Trials 2-8 lie within 8.6% of the average of those values, with the exception of one value being 12% away. Figure 3 shows the variability of the results returned by the PSO algorithm. For Trials 1-2, 1500 iterations may not have been enough, and the code may have stagnated at a local minimum $J$-value. This means that the pattern-deviating results in Figure 2 could simply be the result of this variability and not the inability of the algorithm to determine the correct trajectory.
The base code has a run time of approximately 17 minutes. The ellipse code, on the other hand, has a run time ranging from 25-30 minutes. Although this is not an especially long interval of time, a spacecraft’s orbital transfers may be time sensitive and unable to wait 30 minutes for the calculations to be completed. The non-coaxial code’s run time varies depending upon the eccentricity of the orbit. For eccentricities less than 0.3, the run time is approximately 30 minutes. The run time increases up to roughly 90 minutes for an eccentricity greater than 0.8. This is not an acceptable length of time for determining the transfer trajectory of a spacecraft, and measures must be taken to decrease this run time.
5. Conclusions and Future Work

The particle swarm optimization technique is successful at determining the optimal (in regards to propellant efficiency) transfer between orbits. The PSO algorithm produces better results when the constraints on the objective function are based on the orbits’ characteristics rather than the spacecraft’s motion. The accuracy of the algorithm, however, is strongly linked to the penalties and penalty conditions placed upon the constraints.

Based on this research, it is possible to apply the PSO theory to non-coaxial elliptical orbits. The accuracy of the results is unknown, but the results follow the expected trend, and multiple runs of the same initial conditions return values with relatively low deviations.

In the future, the PSO technique can be adjusted for different applications, such as the transfer between two elliptical orbits with different eccentricities, both coaxial and non-coaxial. It can also be applied to rendezvous problems and interplanetary trajectories. There are many possible applications of the PSO algorithm to low-thrust orbital transfers; however, further research should be done to develop an algorithm to calculate the optimal penalty/penalty condition combination.

As the code has a long run time, it would be beneficial to convert this stochastic method into a hybrid one, combining both stochastic and deterministic algorithms. This would hopefully decrease the error and substantially shorten the run time.
References


Appendix A: Ellipse MATLAB Code

A.1 Main Code

A.1.1 PSO_Ellipses.m

```matlab
%% PSO With Constant Accelerator Coefficients

clear all;
global P J JBest PBest GG N_particles N_elements V BLv BUv BLp BUp
global N_iterations GBest
global tI t2
global mass_ratio mass_ratio_Hohmann

N_particles = 50; N_elements = 12; N_iterations = 2000;

particles = 1:N_particles;

%% create random initial population
P = zeros(N_particles, N_elements);
P(:,1:8) = -1*ones(N_particles, 8) + 2*rand(N_particles, 8);
P(:,9) = -pi/2*ones(N_particles, 1) + pi/2*rand(N_particles, 1);
P(:,10) = 3*rand(N_particles, 1);
P(:,11) = 2*pi*rand(N_particles, 1);
P(:,12) = 3*rand(N_particles, 1);

PBest = zeros(N_particles, N_elements);
J = zeros(N_particles, 1);
JBest = zeros(N_particles, 1);
V = zeros(N_particles, N_elements);

%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1 -1 -1 -1 -1 -1 -1 -pi/2 1e-6 0 1e-6];
BUp = [1 1 1 1 1 1 1 1 pi/2 3 2*pi 3];

%% determine velocity bounds
BUv = BUp - BLp;
BLv = -BUv;

JBest = zeros(N_particles, 1);
for i = 1:N_particles
    JBest(i) = inf;
end
GG = inf;

for j = 1:N_iterations
    fprintf('Iteration number: %g \n', j)
    EvalJ_Ellipse;
    EvalPGBest_Ellipse;
    UpdateV_Ellipse(j);
    UpdateP_Ellipse;
    GGstar(j) = GG;
    fprintf('GG = %6.8f \n', GG)
    fprintf('dt1 = %6.8f, dt2 = %6.8f \n', GBest(10), GBest(12))
    fprintf('theta0 = %g \n', GBest(9))
```

19
fprintf('mf/m0 = %6.8f \n', mass_ratio)

N_change = 25;
if j==50 || j==100 || j==150 || j==200 || j==250 || j==300 || j==350 ... || j==400 || j==450 || j==550 || j==650 || j==750 || j==850 || j==950
indices = j-10:j;
space = length(indices);
for f=1:space
    terms(f) = GGstar(indices(f));
end
avg = sum(terms)/length(terms);
test = abs((GGstar(j) - avg)/GGstar(j))*100;
if test < .01
    disp('P matrix is redefined.' )
P(1:N_change,1:8) = -1*ones(N_change,8) + 2*rand(N_change,8);
P(1:N_change,9) = -pi/2*ones(N_change,1) + pi/2*rand(N_change,1);
P(1:N_change,10) = 3*rand(N_change,1);
P(1:N_change,11) = 2*pi*rand(N_change,1);
P(1:N_change,12) = 3*rand(N_change,1);
else
    disp('P matrix is not redefined.' )
end
end
iterations = [1:N_iterations];
a = log(GGstar); figure
plot(iterations, a) xlabel('Iteration Number') ylabel('ln(GGstar)')
save 'PSO_adapt' GGstar, N_iterations

A.2 Functions

A.2.1 EvalJ_Ellipse.m

function EvalJ_Ellipse()

% EvalJ evaluates J for each particle in current iteration

global P J N_particles
global t1 t2
global zeta0 zeta1 zeta2 zeta3 s_theta0 s_theta1 s_theta2 s_theta3 delta_t1 deltaE delta_t2

% Initialize values
muB = 1; %DU^3/TU^2
c = .5; %DU/TU
n0 = .2; %DU/TU^2
beta = 2;
a1 = 1;
a2 = beta*a1;
ecc1 = 0.0;
ecc2 = 0.0;
energy1 = -muB/(2*a1);
energy2 = -muB/(2*a2);
%% Start calculations to calculate J
for i = 1:N_particles
    zeta0 = P(i,1);
    zeta1 = P(i,2);
    zeta2 = P(i,3);
    zeta3 = P(i,4);
    s_theta0 = P(i,5);
    s_theta1 = P(i,6);
    s_theta2 = P(i,7);
    s_theta3 = P(i,8);
    theta0 = P(i,9);
    delta_t1 = P(i,10);
    deltaE = P(i,11);
    delta_t2 = P(i,12);

    \begin{align*}
    p1 &= a1(1 - ecc1^2); \\
    r0 &= \sqrt{2*(\text{energy1} + \mu_B/r0)}; \\
    v_r0 &= \sqrt{(\mu_B/(a1*(1-ecc1^2)))*ecc1*sin(\theta0)}; \\
    v_theta0 &= \sqrt{(\mu_B/(a1*(1-ecc1^2)))*(1 + ecc1*cos(\theta0))}; \\
    \theta0 &= \frac{v_theta0}{r0}; \\
    \mathbf{x}_0 &= \begin{bmatrix} r0; v_r0; \theta0; \theta0 \end{bmatrix}; \\
    \text{tspan} &= [0 \delta t_1]; \\
    \text{options} &= \text{odeset}('\text{RelTol}', 1e-8, '\text{'AbsTol}', 1e-8); \\
    [t,\mathbf{x}] &= \text{ode45}('\text{PSOode}', \text{tspan}, \mathbf{x}_0, \text{options}); \\
    \text{endlimit} &= \text{length}(\mathbf{x});
\end{align*}

%% Initial values for Keplerian coast arc
    \text{t1} &= \delta t_1; \\
    \text{r1} &= x(\text{endlimit},1); \\
    \text{v_r1} &= x(\text{endlimit},2); \\
    \text{theta1} &= x(\text{endlimit},3); \\
    \text{theta_dot1} &= x(\text{endlimit},4); \\
    \text{v_theta1} &= r1*\theta0; \\
    \text{energy_coast} &= v1^2/2 - \mu_B/r1; \\
    \text{a_coast} &= (\mu_B*r1)/(2*\mu_B - r1*(v_r1^2 + v_theta1^2)); \\
    \text{ecc_coast} &= \sqrt{(1 - (r1^2*v_theta1^2))/(\mu_B*a_coast)); \\
    \text{E1} &= 2*atan( \sqrt{((1-ecc_coast)/(1+ecc_coast))}*tan(\theta1/2) ); \\
    \text{if} \quad E1 < 0 \\
    \quad E1 &= E1 + 2*pi; \\
    \end{align*}
\text{E2} &= E1 + \delta E; \\
    \text{theta2} &= 2*atan( \sqrt{((1+ecc_coast)/(1-ecc_coast))}*tan(E2/2) ); \\
    \text{if} \quad \text{theta2} < 0 \\
    \quad \text{theta2} &= \text{theta2} + 2*pi; \\
    \end{align*}
\text{coast_time} &= \sqrt{(a_coast^3/\mu_B)*(\delta E - ecc_coast*(\sin(E2) - \sin(E1))};
}\%
\text{Initial values for second thrust arc}
\text{v_r2} &= \sqrt{(\mu_B/(a_coast*(1-ecc_coast^2)))\ast ecc_coast*\sin(\theta2)}; \\
\text{v_theta2} &= \sqrt{(\mu_B/(a_coast*(1-ecc_coast^2)))\ast(1+ecc_coast*\cos(\theta2))}; \\
\text{r2} &= a_coast*(1-ecc_coast^2)/(1+ ecc_coast*\cos(\theta2)); \\
\text{xi2} &= x1 + (\text{theta2} - \text{theta1}); \\
\text{theta_dot2} &= \text{v_theta2}/r2; \\
\text{x2} &= \begin{bmatrix} r2; v_r2; \theta2; \theta2 \end{bmatrix};
t2 = delta_t1 + coast_time;
 tf = t2 + delta_t2;

tspan2 = [t2 tf];
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x] = ode45('PSOode2',tspan2,x2,options);
endlimit = length(x(:,1));

rf = x(endlimit,1);
v_rf = x(endlimit,2);
thetaf = x(endlimit,3);
theta_dotf = x(endlimit,4);

v_thetaf = rf*theta_dotf;
vf = sqrt(v_rf^2 + v_thetaf^2);
a2_calc = muB*rf/( 2*muB - rf*(v_rf^2 + v_thetaf^2));

phi = thetaf; % angle wrt original LOA

Rf = [rf 0 0];
Vf = [v_rf v_thetaf 0];
hf = norm(cross(Rf,Vf));
pf = hf^2/muB;
energyf = vf^2/2 - muB/rf;
ecc2_calc = sqrt(1 + 2*pf*energyf/muB);

if v_rf > 0
 theta_new = acos( (1/ecc2_calc)*(pf/rf - 1) ); % angle with respect to
 new LOA (should be the same)
else
 theta_new = 2*pi + acos( (1/ecc2_calc)*(pf/rf - 1) );
end

%% Calculate J

for j=1:4
 if abs(d(j)) < 1e-3
 alpha(j) = 0;
 else
 alpha(j) = 150;
 end
end
\[ K(j) = \text{abs}(d(j)) \times \alpha(j); \]
\[ J(i) = \delta_{t1} + \delta_{t2} + \sum(K); \]

**A.2.2 EvalPGBest_Ellipse.m**

```matlab
function EvalPGBest_Ellipse()
  global P J JBest PBest GG GBest N_particles
  global mass_ratio mass_ratio_Hohmann
  c = .5;
  n0 = .2;
  for i = 1:N_particles
    if J(i) < JBest(i)
      PBest(i,:) = P(i,:);
      JBest(i) = J(i);
    end
  end
  for i = 1:N_particles
    if J(i) < GG
      GG = J(i);
      GBest = P(i,:);
      mass_ratio = 1 - (n0/c)*(GBest(10) + GBest(12));
    end
  end
end
```

**A.2.3 PSOode.m**

```matlab
function [ xdot ] = PSOode( t,x )
  global zeta0 zeta1 zeta2 zeta3
  xdot = zeros(4,1);
  muB = 1;
  c = .5;
  n0 = .2;
  T/m = (c*n0)/(c - n0*t);
  delta = zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3;
  xdot(1) = x(2);
  xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 + ((c*n0)/(c - n0*t))*sin(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3);
  xdot(3) = x(4);
  xdot(4) = (-2*x(2)*x(4) + ((c*n0)/(c - n0*t))*cos(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3))/x(1);
end
```
A.2.4 PSOode2.m

```matlab
function [ xdot ] = PSOode2( t,x )
%% State variables for PSO algorithm for second thrust arc
% t2 < t < tf

global t1 t2
global s_theta0 s_theta1 s_theta2 s_theta3

xdot = zeros(4,1);

muB = 1;
c = .5;
n0 = .2;
% T/m = (c*n0)/(c - n0*(t1 + t - t2));
% delta = s_theta0 + s_theta1*(t - t2) + s_theta2*(t - t2)^2 + s_theta3*(t - t2)^3;

xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/(x(1)^2) + ((c*n0)/(c - n0*(t1+t-t2)))*sin(s_theta0 + s_theta1*(t - t2) + s_theta2*(t - t2)^2 + s_theta3*(t - t2)^3);

xdot(3) = x(4);
xdot(4) = (-2*x(2)*x(4) + ((c*n0)/(c - n0*(t1 + t - t2)))*cos(s_theta0 + s_theta1*(t-t2) + s_theta2*(t-t2)^2 + s_theta3*(t-t2)^3))/x(1);

end
```

A.2.5 UpdateP_Ellipse.m

```matlab
function UpdateP_Ellipse()
%% UpdateP updates the position vector

global P N_particles N_elements V BLp BUp

for i =1:N_particles
    P(i,:) = P(i,:) + V(i,:);
    for k = 1:N_elements
        if P(i,k) < BLp(k)
            P(i,k) = BLp(k);
            V(i,k) = 0;
        end
        if P(i,k) > BUp(k)
            P(i,k) = BUp(k);
            V(i,k) = 0;
        end
    end
end
```

A.2.6 UpdateV_Ellipse.m

```matlab
function UpdateV_Ellipse(j)
%% UpdateV updates the velocity vector V
%% Variable accelerator coeffs.
```
global P PBest GBest N_particles N_elements V BLv BUv
global N_iterations

c_I = (1 + rand)/2;
c_C = 1.49445*rand;
c_S = 1.49445*rand;

for i =1:N_particles
    V(i,:) = c_I*V(i,:) + c_C*(PBest(i,:) - P(i,:)) + c_S*(GBest - P(i,:));
    for k = 1:N_elements
        if V(i,k) < BLv(k)
            V(i,k) = BLv(k);
        end
        if V(i,k) > BUv(k)
            V(i,k) = BUv(k);
        end
    end
end
Appendix B: Base MATLAB Code

B.1 Main Code

B.1.2 PSO_base.m

```matlab
%% PSO With Variable Accelerator Coefficients
% clc;
clear all;
global P J JBest PBest GG N_particles N_elements V BLv BUv BLp BUp
global N_iterations GBest
global t1 t2
global mass_ratio mass_ratio_Hohmann

N_particles = 50;
N_elements = 11;
N_iterations = 2000;

%% create random initial population
P = zeros(N_particles, N_elements);
P(:,1:8) = -1*ones(N_particles, 8) + 2*rand(N_particles, 8);
P(:,9) = 3*rand(N_particles, 1);
P(:,10) = 2*pi*rand(N_particles, 1);
P(:,11) = 3*rand(N_particles, 1);

PBest = zeros(N_particles, N_elements);
J = zeros(N_particles);
JBest = zeros(N_particles);
V = zeros(N_particles, N_elements);

%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1 -1 -1 -1 -1 -1 -1 -1 1e-6 1e-6];
BUp = [1 1 1 1 1 1 1 3 2*pi 3];

%% determine velocity bounds
BUv = BUv - BLp;
BLv = -BUv;
for i = 1:N_particles
    JBest(i,1) = inf;
end
GG = inf;

for j = 1:N_iterations
    fprintf('Iteration number: %g \n', j)
    EvalJ_base;
    EvalPGBest_base;
    UpdateV_base(j);
    UpdateP_base;
    if j==100 || j==200 || j==300 || j==400 || j==500 || j==600 || j==700 ||
        j==800 || j==900
        GG = 2*GG; % Re-initiates J to check for stagnation
    end
    GGstar(j) = GG;
    fprintf('GG = %6.8f \n', GG)
    fprintf('dt1 = %6.8f, dt2 = %6.8f \n', GBest(9), GBest(11))
    fprintf('mf/m0 = %6.8f \n', mass_ratio)
end
```

```
iterations = [1:N_iterations];
a = log(GGstar);
plot(iterations, a)
xlabel('Iteration Number')
ylabel('ln(GGstar)')
save 'PSO_adapt' GGstar, N_iterations

B.2 Functions

B.2.1 EvalJ_base.m

function EvalJ_base()
% EvalJ evaluates J for each particle in current iteration

global P J N_particles
global t1 t2
global zeta0 zeta1 zeta2 zeta3 s_theta0 s_theta1 s_theta2 s_theta3 delta_t1
    deltaE delta_t2

% Initialize values
muB = 1; % DU^3/TU^2
R1 = 1;
c = .5; % DU/TU
n0 = .2; % DU/TU^2
beta = 10;
R2 = R1*beta;

v_r0 = 0;
v_theta0 = sqrt(muB/R1);
r0 = R1;
xi0 = 0;
theta0 = 0;
theta_dot0 = v_theta0/r0;

% Start calculations to calculate J
for i = 1:N_particles
    zeta0 = P(i,1);
    zeta1 = P(i,2);
    zeta2 = P(i,3);
    zeta3 = P(i,4);
    s_theta0 = P(i,5);
    s_theta1 = P(i,6);
    s_theta2 = P(i,7);
    s_theta3 = P(i,8);
    delta_t1 = P(i,9);
    deltaE = P(i,10);
    delta_t2 = P(i,11);

    x0 = [r0; v_r0; theta0; theta_dot0];
    tspan = [0 delta_t1];
    options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
    [t,x] = ode45('PSOode', tspan, x0, options);
    endlimit = length(x);
%% Initial values for Keplerian coast arc
\begin{verbatim}
t1 = delta_t1;
r1 = x(endlimit,1);
v_r1 = x(endlimit,2);
theta1 = x(endlimit,3);
theta_dot1 = x(endlimit,4);

v_theta1 = r1*theta_dot1;
v1 = sqrt(v_r1^2 + (r1*theta_dot1)^2);
xi1 = v1^2/2 - muB/r1;
a = (muB*r1)/(2*muB - r1*(v_r1^2 + v_theta1^2));
ecc = sqrt( 1 - (r1^2*v_theta1^2)/(muB*a));
E1 = 2*atan( sqrt((1-ecc)/(1+ecc)) * tan(theta1/2) ); %Use tangent relation 

\text{b/t theta and E}
\begin{verbatim}
  if E1 < 0
    E1 = E1 + 2*pi;
  end
  f1 = 2*atan( sqrt((1+ecc)/(1-ecc)) * tan(E1/2) );
  if f1 < 0
    f1 = f1 + 2*pi;
  end
E2 = E1 + deltaE;
f2 = 2*atan( sqrt((1+ecc)/(1-ecc)) * tan(E2/2) );
  if f2 < 0
    f2 = f2 + 2*pi;
  end
theta2 = f2;
coast_time = sqrt(a^3/muB)*(deltaE - ecc*(sin(E2) - sin(E1)));
\end{verbatim}
\end{verbatim}
%% Initial values for second thrust arc
\begin{verbatim}
v_r2 = sqrt(muB/(a*(1-ecc^2)))*ecc*sin(f2);
v_theta2 = sqrt(muB/(a*(1-ecc^2)))*(1+ecc*cos(f2));
r2^2 = a*(1-ecc^2)/(1+ecc*cos(f2));
xi2 = xi1 + (f2 - f1);
theta_dot2 = v_theta2/r2;
x2 = [r2; v_r2; theta2; theta_dot2];
t2 = delta_E1 + coast_time;
tf = t2 + delta_t2;
tspan2 = [t2 tf];
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x] = ode45('PSOode2',tspan2,x2,options);
endlimit = length(x(:,1));
rf = x(endlimit,1);
v_rf = x(endlimit,2);
thetaf = x(endlimit,3);
theta_dotf = x(endlimit,4);

v_thetaf = rf*theta_dotf;
\end{verbatim}
%% Calculate J
\begin{verbatim}
d = zeros(3,1);
d(1) = v_rf;
d(2) = v_thetaf - sqrt(muB/R2);
d(3) = rf - R2;
\end{verbatim}
\end{verbatim}
\end{verbatim}
for j=1:3
\begin{verbatim}
if abs(d(j)) < 1e-3
    alpha(j) = 0;
else
    alpha(j) = 100;
end

K(j) = abs(d(j))*alpha(j);
end

J(i) = delta_t1 + delta_t2 + sum(K);
end

B.2.2 EvalPGBest_base.m

function EvalPGBest_base()
    global P J JBest PBest GG GBest N_particles
global mass_ratio mass_ratio_Hohmann

c = .5;
n0 = .2;

for i = 1:N_particles
    if J(i) < JBest(i)
        PBest(i,:) = P(i,:);
        JBest(i) = J(i);
    end
end

for i = 1:N_particles
    if J(i) < GG
        GG = J(i);
        GBest = P(i,:);
        mass_ratio = 1 - (n0/c)*(GBest(9) + GBest(11));
    end
end

B.2.3 PSOode.m

function [ xdot ] = PSOode2( t,x )
    global t1 t2
    global s_theta0 s_theta1 s_theta2 s_theta3

ddot = zeros(4,1);

muB = 1;
c = .5;
n0 = .2;
\end{verbatim}
% \( \frac{T}{m} = \frac{(c \cdot n_0)}{(c - n_0 \cdot (t_1 + t - t_2))} \);
% delta = \( s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3 \);

\[
\begin{align*}
    x_{dot}(1) &= x(2); \\
    x_{dot}(2) &= (-\mu_B + x(1) \cdot (x(1) \cdot x(4))^2)/(x(1)^2) + ((c \cdot n_0)/(c - n_0 \cdot (t_1 + t - t_2))) \cdot \sin(s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3); \\
    x_{dot}(3) &= x(4); \\
    x_{dot}(4) &= (-2 \cdot x(2) \cdot x(4) + ((c \cdot n_0)/(c - n_0 \cdot (t_1 + t - t_2))) \cdot \cos(s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3))/x(1);
\end{align*}
\]

end

B.2.4 PSOode2.m

\[
\begin{align*}
    \text{function } [ xdot ] &= \text{PSOode2}( t, x ) \\
    \% \text{State variables for PSO algorithm for second thrust arc} \\
    \% t_2 < t < t_f \\
    \text{global } t_1 t_2 \\
    \text{global } s_{\theta 0} s_{\theta 1} s_{\theta 2} s_{\theta 3} \\
    xdot &= \text{zeros}(4,1); \\
    \mu_B &= 1; \\
    c &= .5; \\
    n_0 &= .2; \\
    \% \frac{T}{m} = \frac{(c \cdot n_0)}{(c - n_0 \cdot (t_1 + t - t_2))} \\
    \% \text{delta} = \text{delta} \text{ of } s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3; \\
    xdot(1) &= x(2); \\
    xdot(2) &= (-\mu_B + x(1) \cdot (x(1) \cdot x(4))^2)/(x(1)^2) + ((c \cdot n_0)/(c - n_0 \cdot (t_1 + t - t_2))) \cdot \sin(s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3); \\
    xdot(3) &= x(4); \\
    xdot(4) &= (-2 \cdot x(2) \cdot x(4) + ((c \cdot n_0)/(c - n_0 \cdot (t_1 + t - t_2))) \cdot \cos(s_{\theta 0} + s_{\theta 1} \cdot (t - t_2) + s_{\theta 2} \cdot (t - t_2)^2 + s_{\theta 3} \cdot (t - t_2)^3))/x(1);
\end{align*}
\]

end

B.2.5 UpdateP_base.m

\[
\begin{align*}
    \text{function } \text{UpdateP_base}() \\
    \% \text{UpdateP updates the position vector} \\
    \text{global } P N_{\text{particles}} N_{\text{elements}} V BLp BUp \\
    \text{for } i = 1: N_{\text{particles}} \\
    \quad P(i,:) = P(i,:) + V(i,:); \\
    \text{for } k = 1: N_{\text{elements}} \\
    \quad \text{if } P(i,k) < \text{BLp}(k) \\
    \quad \quad P(i,k) = \text{BLp}(k); \\
    \quad \quad V(i,k) = 0; \\
    \quad \text{end} \\
    \quad \text{if } P(i,k) > \text{BUp}(k) \\
    \quad \quad P(i,k) = \text{BUp}(k); \\
    \quad \quad V(i,k) = 0; \\
\end{align*}
\]
B.2.6 UpdateV_base.m

function UpdateV_base(j)

% UpdateV updates the velocity vector V
% Variable accelerator coeffs.

global P PBest GBest N_particles N_elements V BLv BUv
global N_iterations

c_I = (1 + rand)/2;
c_C = 1.49445*rand;
c_S = 1.49445*rand;

for i =1:N_particles
    V(i,:) = c_I*V(i,:) + c_C*(PBest(i,:) - P(i,:)) + c_S*(GBest - P(i,:));
    for k = 1:N_elements
        if V(i,k) < BLv(k)
            V(i,k) = BLv(k);
        end
        if V(i,k) > BUv(k)
            V(i,k) = BUv(k);
        end
    end
end
Appendix C: Non-coaxial MATLAB Code

C.1 EvalJ_nonax.m

function EvalJ_Ellipse()
% EvalJ evaluates J for each particle in current iteration

global P J N_particles
global t1 t2
global zeta0 zeta1 zeta2 zeta3 s_theta0 s_theta1 s_theta2 s_theta3 delta_t1
deltaE delta_t2
global ecc1 ecc2 a1 a2 thetaf theta_new phi alpha

%% Initialize values
muB = 1; %DU^3/TU^2
c = .5; %DU/TU
n0 = .2; %DU/TU^2
beta = 2;
a1 = 1;
a2 = beta*a1;
psi = pi/2;
ecc1 = 0.9;
ecc2 = 0.9;
energy1 = -muB/(2*a1);
energy2 = -muB/(2*a2);

%% Start calculations to calculate J
for i = 1:N_particles
% fprintf('particle no: %g',i)
zeta0 = P(i,1);
zeta1 = P(i,2);
zeta2 = P(i,3);
zeta3 = P(i,4);
s_theta0 = P(i,5);
s_theta1 = P(i,6);
s_theta2 = P(i,7);
s_theta3 = P(i,8);
theta0 = P(i,9);
delta_t1 = P(i,10);
deltaE = P(i,11);
delta_t2 = P(i,12);
p1 = a1*(1 - ecc1^2);
r0 = p1/(1 + ecc1*cos(theta0));
v0 = sqrt(2*(energy1 + muB/r0));
v_r0 = sqrt( muB/(a1*(1-ecc1^2)))*ecc1*sin(theta0); % equation taken from paper
v_theta0 = sqrt( muB/(a1*(1-ecc1^2)))*(1 + ecc1*cos(theta0));
theta_dot0 = v_theta0/r0;
x0 = [r0; v_r0; theta0; theta_dot0];
tspan = [0 delta_t1];
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x] = ode45('PSOode', tspan, x0, options);
endlimit = length(x);

% Initial values for Keplerian coast arc
% t1 = delta_t1;
r1 = x(endlimit,1);
v_r1 = x(endlimit,2);
theta1 = x(endlimit,3);
theta_dot1 = x(endlimit,4);

v_theta1 = r1*theta_dot1;

v1 = sqrt(v_r1^2 + (r1*theta_dot1)^2);
x1 = v_theta1/r1;
energy_coast = v1^2/2 - muB/r1;

a_coast = (muB*r1)/(2*muB - r1*(v_r1^2 + v_theta1^2));
ecc_coast = sqrt(1 - (r1^2*v_r1^2)/(muB*a_coast));
E1 = 2*atan( sqrt((1-ecc_coast)/(1+ecc_coast)) * tan(theta1/2) );
    %Use tangent relation b/t theta and E
    if E1 < 0
        E1 = E1 + 2*pi;
    end
E2 = E1 + deltaE;
theta2 = 2*atan( sqrt((1+ecc_coast)/(1-ecc_coast)) * tan(E2/2) );
    if theta2 < 0
        theta2 = theta2 + 2*pi;
    end
coast_time = sqrt(a_coast^3/muB)*(deltaE - ecc_coast*(sin(E2) - sin(E1)));

% Initial values for second thrust arc
v_r2 = sqrt(muB/(a_coast*(1-ecc_coast^2)))*ecc_coast*sin(theta2);
v_theta2 = sqrt(muB/(a_coast*(1-ecc_coast^2)))*(1+ecc_coast*cos(theta2));
r2 = a_coast*(1-ecc_coast^2)/(1+ecc_coast*cos(theta2));
x2 = x1 + (theta2 - theta1);
theta_dot2 = v_theta2/r2;

x2 = [r2; v_r2; theta2; theta_dot2];
t2 = delta_E1 + coast_time;
tf = t2 + delta_t2;
tspan2 = [t2 tf];
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
[t,x] = ode45('PSOode2',tspan2,x2,options);
endlimit = length(x(:,1));

rf = x(endlimit,1);
v_rf = x(endlimit,2);
thetaf = x(endlimit,3);
theta_dotf = x(endlimit,4);

v_thetaf = rf*theta_dotf;
vf = sqrt(v_rf^2 + v_thetaf^2);
a2_calc = muB*rf/( 2*muB - rf*(v_rf^2 + v_thetaf^2));
phi = thetaf;    % angle wrt original LOA

Rf = [rf 0 0];
Vf = [v_rf v_thetaf 0];
hf = norm( cross(Rf,Vf) );
pf = hf^2/muB;
energyf = vf^2/2 - muB/rf;
ecc2_calc = sqrt(1 + 2*pf*energyf/muB);
    if v_rf > 0
        theta_new = acos( (1/ecc2_calc)*(pf/rf - 1) );
    else
        theta_new = 2*pi + acos( (1/ecc2_calc)*(pf/rf - 1) );
    end
%% Calculate J

d = zeros(3,1);
d(1) = phi - theta_new - psi;
d(2) = a2_calc - a2;
d(3) = ecc2_calc - ecc2;

for j=1:3
    if abs(d(j)) < 1e-3
        alpha(j) = 0;
    else
        alpha(j) = 100;
    end

    if abs(d(j)) < 1e-3
        alpha(j) = 0;
    else
        alpha(j) = 25;
    end

    if abs(d(j)) < 1e-3
        alpha(j) = 0;
    else
        alpha(j) = 25;
    end

    K(j) = abs(d(j))*alpha(j);
end

J(i) = delta_t1 + delta_t2 + sum(K);
end
Academic Vita of Kaomi C. Hashimoto

Address: 266 Simmons Hall
University Park, PA 16802
E-mail: kaomihashimoto@gmail.com

EDUCATION

Bachelor of Science in Aerospace Engineering, May 2012
The Pennsylvania State University
Schreyer Honors College
- Honors in Aerospace Engineering
Thesis title: An Evolutionary Algorithm for Low-Thrust Orbital Transfers
Thesis Supervisor: Dr. Robert G. Melton

RELATED EXPERIENCE

Engineering Intern, The Boeing Company, Long Beach, CA
Reliability, Maintainability, & Availability of the C-17 Globemaster III
May 2011-August 2011

Research Intern, Carnegie Mellon University, Pittsburgh, PA
NASA Summer Undergraduate Research Program
Department of Physics
June 2010-July 2010

Lead Engineering Intern, Honeywell Aerospace, Phoenix, AZ
Supplier Quality Assurance
August 2007-August 2008

PRESENTATIONS

Title: Particle Swarm Optimization Applied to Astrodynamics
Conference: AIAA Region I-MA Student Conference
Date: April 14, 2012

AWARDS/HONORS

The Boeing Company Scholarship
Paul Morrow Endowed Scholarship
Schreyer Honors College Academic Excellence Scholarship
MEPO Leadership Award
Dean’s List (7 of 8 semesters)

ACTIVITIES

Vice President, Sigma Gamma Tau
Current Member Liaison, Tau Beta Pi
Member, American Institute for Aeronautics and Astronautics
Member, Society of Hispanic Professional Engineers