SPACECRAFT TRAJECTORY OPTIMIZATION SUITE (STOPS):
OPTIMIZATION OF LOW-THRUST INTERPLANETARY
SPACECRAFT TRAJECTORIES USING MODERN
OPTIMIZATION TECHNIQUES

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

by
Shane P. Sheehan
September 2017
COMMITTEE MEMBERSHIP


AUTHOR: Shane P. Sheehan

DATE SUBMITTED: September 2017

COMMITTEE CHAIR: Kira Abercromby, Ph.D.
Associate Professor of Aerospace Engineering

COMMITTEE MEMBER: Eric Mehiel, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Jordi Puig-Suari, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Colleen Kirk, Ph.D.
Professor of Mathematics
ABSTRACT


Shane P. Sheehan

The work presented here is a continuation of Spacecraft Trajectory Optimization Suite (STOpS), a master's thesis written by Timothy Fitzgerald at California Polytechnic State University, San Luis Obispo. Low-thrust spacecraft engines are becoming much more common due to their high efficiency, especially for interplanetary trajectories. The version of STOpS presented here optimizes low-thrust trajectories using the Island Model Paradigm with three stochastic evolutionary algorithms: the genetic algorithm, differential evolution, and particle swarm optimization. While the algorithms used here were designed for the original STOpS, they were modified for this work.

The low-thrust STOpS was successfully validated with two trajectory problems and their known near-optimal solutions. The first verification case was a constant-thrust, variable-time Earth orbit to Mars orbit transfer where the thrust was 3.787 Newtons and the time was around approximately 195 days. The second verification case was a variable-thrust, constant-time Earth orbit to Mercury orbit transfer with the thrust coming from a solar electric propulsion model equation and the time being 355 days. Low-thrust STOpS found similar near-optimal solutions in each case. The final result of this work is a versatile MATLAB tool for optimizing low-thrust interplanetary trajectories.
ACKNOWLEDGMENTS

I would like to thank Dr. Kira Abercromby for her support and assistance throughout this project. Despite the fact that she was abroad during the majority of the time that I was working my thesis work, I would receive periodic unsolicited texts or emails to keep me focused and on track. Without her, I would have had a much harder time meeting my timeline.

I would also like to thank Timothy Fitzgerald for creating such a fantastic work as a foundation for my thesis. Obviously, this thesis wouldn’t have been possible without the countless hours he put into STOpS to solve impulse-thrust trajectory problems. Adapting his algorithms and overall structure allowed me to solve the fundamentally different problem of low-thrust orbit transfers.

I would like to thank Andrew Blocher for pushing me to start my thesis and meet all the deadlines. Somehow, we ended up on the same exact timeline for finishing our theses and completing school. Keeping pace with each other helped us both make it to the finish line.

Finally, I would like to thank my family for supporting me throughout the extra time I took to finish my schooling and for putting up with me living at home for five weeks while I worked on my thesis all day and complained all night. There were certain times that I wasn’t sure I had chosen the right topic for my thesis, but they encouraged me to stick it through to the end and I’m glad I did.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ ix
LIST OF FIGURES ..................................................................................................... xi
CHAPTER 1. INTRODUCTION ................................................................................ 1
  1.1 Statement of Problem ..................................................................................... 1
  1.2 Purpose of Study ........................................................................................... 2
  1.3 Current Optimization Suites ......................................................................... 3
  1.4 Structure of Paper ......................................................................................... 7
  1.5 Acronyms ........................................................................................................ 8
CHAPTER 2. ORBITAL MECHANICS ...................................................................... 9
  2.1 High-Thrust Versus Low-Thrust Engines ....................................................... 9
  2.2 High-Thrust Orbit Transfer Mechanics ......................................................... 10
  2.3 Low-Thrust Orbit Transfer Mechanics ......................................................... 11
CHAPTER 3. OPTIMIZATION SETUP ..................................................................... 14
  3.1 High-Thrust Trajectory Variable Reduction .................................................. 14
  3.2 Low-Thrust Problem Setup ........................................................................... 16
    3.2.1 2D Polar Coordinates ............................................................................. 16
    3.2.2 Canonical Units ....................................................................................... 17
    3.2.3 Propagation and Equations of Motion ................................................... 18
    3.2.4 Cost Function .......................................................................................... 19
  3.3 Low-Thrust Trajectory Variable Reduction .................................................. 22
    3.3.1 The Segmented Method ......................................................................... 23
    3.3.2 The Costate Method ............................................................................... 24
  3.4 User Interface .................................................................................................. 28
CHAPTER 4. OPTIMIZATION ................................................................................. 30
  4.1 Local Versus Global Optimum ....................................................................... 31
  4.2 Genetic Algorithm ........................................................................................ 32
    4.2.1 Binary Versus Continuous Genetic Algorithms ..................................... 33
    4.2.2 Selection Methods .................................................................................. 34
    4.2.3 Mating Methods ..................................................................................... 37
    4.2.4 Mutation ................................................................................................ 41
| A-1 | Departure Planet and Arrival Planet | ......................................................... | 84 |
| A-2 | Island Model Paradigm Options | ................................................................... | 85 |
| A-3 | Genetic Algorithm Parameters | ................................................................... | 87 |
| A-4 | Differential Evolution Parameters | ........................................................... | 89 |
| A-5 | Particle Swarm Optimization Parameters | ................................................ | 90 |
| A-6 | Cost Function Selection and Parameters | ................................................. | 92 |
| A-7 | Low-Thrust Mission Parameters | ................................................................... | 94 |
| A-8 | Earliest and Latest Departure Date | ................................................................... | 96 |
| B-1 | Thrust Profile Plots | ................................................................... | 96 |
| B-2 | Test Case 1 | ................................................................... | 97 |
| B-3 | Test Case 2 | ................................................................... | 98 |
| B-3 | Earth-to-Jupiter Test Case | ................................................................... | 99 |
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. Description of JPL Low-Thrust Trajectory Tools</td>
<td>4</td>
</tr>
<tr>
<td>Table 2. Availability of JPL Low-Thrust Trajectory Tools</td>
<td>5</td>
</tr>
<tr>
<td>Table 3. Island Model Parameters for Test Case 1</td>
<td>54</td>
</tr>
<tr>
<td>Table 4. Genetic Algorithm Parameters for Test Case 1</td>
<td>55</td>
</tr>
<tr>
<td>Table 5. Differential Evolution Parameters for Test Case 1</td>
<td>55</td>
</tr>
<tr>
<td>Table 6. Particle Swarm Optimization Parameters for Test Case 1</td>
<td>55</td>
</tr>
<tr>
<td>Table 7. Test Case 1: Comparison of End Conditions for Segmented Method</td>
<td>58</td>
</tr>
<tr>
<td>Table 8. Test Case 1: Comparison of End Conditions for Costate Method</td>
<td>59</td>
</tr>
<tr>
<td>Table 9. Test Case 2: Comparison of End Conditions for Segmented Method</td>
<td>65</td>
</tr>
<tr>
<td>Table 10. Test Case 2: Comparison of End Conditions for Costate Method</td>
<td>67</td>
</tr>
<tr>
<td>Table 11. Earth-to-Jupiter End Condition Comparison for Segmented Method</td>
<td>73</td>
</tr>
<tr>
<td>Table 12. Earth-to-Jupiter End Condition Comparison for Costate Method</td>
<td>74</td>
</tr>
<tr>
<td>Table 13. Island Model Settings Guide</td>
<td>85</td>
</tr>
<tr>
<td>Table 14. Genetic Algorithm Settings Guide</td>
<td>87</td>
</tr>
<tr>
<td>Table 15. Differential Evolution Settings Guide</td>
<td>89</td>
</tr>
<tr>
<td>Table 16. Particle Swarm Optimization Settings Guide</td>
<td>91</td>
</tr>
<tr>
<td>Table 17. Cost Function Settings Guide</td>
<td>93</td>
</tr>
<tr>
<td>Table 18. Low-Thrust Mission Parameters Guide</td>
<td>94</td>
</tr>
</tbody>
</table>

Table 1. Description of JPL Low-Thrust Trajectory Tools 4
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. High-Thrust Orbit Transfer [7]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2. Low-Thrust Orbit Transfer</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3. Thrust Pointing Angle for Low-Thrust Trajectory</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4. 2D Polar Coordinate System</td>
<td>17</td>
</tr>
<tr>
<td>Figure 5. Griewank’s Function</td>
<td>32</td>
</tr>
<tr>
<td>Figure 6. Test Case 1: Valid Trajectory with Segmented Method</td>
<td>57</td>
</tr>
<tr>
<td>Figure 7. Test Case 1: Near-Optimal Solutions [5]</td>
<td>57</td>
</tr>
<tr>
<td>Figure 8. Test Case 1: Valid Trajectory with Costate Method</td>
<td>59</td>
</tr>
<tr>
<td>Figure 9. Test Case 2: Non-Valid Trajectory with Segmented Method</td>
<td>63</td>
</tr>
<tr>
<td>Figure 10. Test Case 2: Valid Trajectory with Segmented Method</td>
<td>65</td>
</tr>
<tr>
<td>Figure 11. Test Case 2: Valid Trajectory with Costate Method</td>
<td>66</td>
</tr>
<tr>
<td>Figure 12. Earth-to-Jupiter Trajectory with Segmented Method</td>
<td>72</td>
</tr>
<tr>
<td>Figure 13. Earth-to-Jupiter Transfer for Costate Method</td>
<td>74</td>
</tr>
<tr>
<td>Figure 14. Departure Planet and Arrival Planet</td>
<td>85</td>
</tr>
<tr>
<td>Figure 15. Island Model Paradigm Options</td>
<td>87</td>
</tr>
<tr>
<td>Figure 16. Genetic Algorithm Parameters</td>
<td>89</td>
</tr>
<tr>
<td>Figure 17. Differential Evolution Parameters</td>
<td>90</td>
</tr>
<tr>
<td>Figure 18. Particle Swarm Optimization Parameters</td>
<td>92</td>
</tr>
<tr>
<td>Figure 19. Cost Function Selection</td>
<td>92</td>
</tr>
<tr>
<td>Figure 20. Cost Function Parameters</td>
<td>94</td>
</tr>
<tr>
<td>Figure 21. Low-Thrust Mission Parameters</td>
<td>96</td>
</tr>
<tr>
<td>Figure 22. Earliest and Latest Departure Date</td>
<td>96</td>
</tr>
<tr>
<td>Figure 23. Test Case 1: Thrust Profile for Segmented Method</td>
<td>97</td>
</tr>
<tr>
<td>Figure 24. Test Case 1: Thrust Profile for Costate Method</td>
<td>97</td>
</tr>
<tr>
<td>Figure 25. Test Case 2: Non-Valid Thrust Profile for Segmented Method</td>
<td>98</td>
</tr>
<tr>
<td>Figure 26. Test Case 2: Valid Thrust Profile for Segmented Method</td>
<td>98</td>
</tr>
<tr>
<td>Figure 27. Test Case 2: Thrust Profile for Costate Method</td>
<td>99</td>
</tr>
<tr>
<td>Figure 28. Earth-to-Jupiter Thrust Profile for Segmented Method</td>
<td>99</td>
</tr>
<tr>
<td>Figure 29. Earth-to-Jupiter Thrust Profile for Costate Method</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 1. High-Thrust Orbit Transfer - 10
CHAPTER 1. INTRODUCTION

1.1 Statement of Problem

Any decent engineering solution involves at least some form of optimization. Optimization can be described as finding the values for a set of variables that will minimize or maximize a certain function. The discovery of calculus introduced optimization. Finding the minima or maxima (optima) for a function was as simple as taking the derivative. Optimization using calculus works well for simple problems where the function is known and there are only a few variables. However, for engineering applications, the problems being optimized tend to be much more complex. Often there are numerous input variables and the function of interest is not known or well defined. When optimizing a complex problem, it is also difficult to tell if the optimum found is truly the best solution (global optima) or just a very good one (local optima).

The traditional definition of spacecraft trajectory optimization involves finding the trajectory that minimizes or maximizes certain parameters. Common parameters that are optimized include the total change in velocity, the transfer time, and final spacecraft mass. This work differs from that definition of optimization in the sense that the trajectory is not being optimized with regard to any specific parameters. What is being minimized is the error of the trajectory’s terminal conditions such as the radius and velocity. By minimizing the error in the terminal conditions this work succeeds in finding a low-thrust trajectory that
sufficiently satisfies the problem. The definition of optimization throughout this work will refer to the minimization of error in end conditions.

Algorithms have been developed specifically to find the global optima for complex problems. As one might suspect, different methods have different strengths and weaknesses, which can make it difficult to choose what method should be used for a particular problem. An alternative to picking one single algorithm is the Island Model Paradigm [7]. The Island Model Paradigm utilizes a suite of different algorithms and allows them to share their best solutions with each other to find the global optima. Allowing multiple algorithms to share their solutions eliminates the need to pick a single method and allows stronger algorithms to compensate for the weakness of certain other algorithms by allowing a different algorithm to pick up the slack.

Timothy Fitzgerald previously created Spacecraft Trajectory Optimization Suite (STOpS) for his thesis at California Polytechnic State University, San Luis Obispo [1]. STOpS is intended to be a publicly available universal spacecraft trajectory optimization tool. It utilizes various modern optimization algorithms and the Island Model Paradigm to optimize trajectories with multiple flybys for cases where the changes in velocity can be considered instantaneous (high-thrust). This work aims to expand STOpS by adding the capability to optimize trajectories for cases where the spacecraft is continuously accelerating throughout the trajectory (low-thrust).

1.2 Purpose of Study
Spacecraft trajectory design is a large part of initial mission planning. The trajectory is one of the biggest design drivers for other satellite subsystems such as power and propulsion. The fuel required to maintain the desired trajectory is a significant portion of a spacecraft’s overall mass and every extra kilogram that must be brought to orbit greatly increases the cost of the mission. For that reason, engineers often spend years designing just one single satellite to make it as light and effective as possible. Therefore, it is advantageous to find a trajectory that requires as little fuel as possible. The desire for highly efficient spacecraft with low fuel mass drove the development of low-thrust engines. As low-thrust spacecraft have become more common, it is increasingly necessary to have a low-thrust trajectory analysis tool for mission planning. This work does not serve to develop new methods or algorithms for trajectory optimization, but to implement available techniques in a practical manner. Other optimization suites do exist, but their availability to the public is limited. The goal of this thesis to develop a tool available to the public that is capable of finding near-optimal low-thrust trajectories for a wide variety of input cases in a reasonable amount of time.

1.3 Literature Current Optimization Suites Review

The low-thrust trajectory optimization suite developed in this work is not the first tool designed for this purpose, but aims to be among the most available, universal, and user friendly. NASA’s Jet Propulsion Laboratory has also developed a suite of tools with a varying degree of fidelity and applications for optimizing low thrust trajectories. The JPL suite includes tools named such as
MALTO, Mystic, Copernicus, OTIS, SNAP, CHEBYTOP, VARITOP, SEPTOP, NEWSEP, and Sail. Overviews of each tool can be found in “Overview of the Development for a Suite of Low-Thrust Trajectory Analysis Tools” [10]. Additional descriptions and details on the availability of each program can be found on the “In-Space Propulsion Technologies Program” section of NASA’s Space Flight Systems webpage [8] and in “Comparison of Performance Predictions for New Low-Thrust Trajectory Tools” [13]. Table 1 shows the general description and fidelity of each JPL tool while Table 2 shows their availability.

Table 1. Description of JPL Low-Thrust Trajectory Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Fidelity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MALTO</td>
<td>Mission Analysis Low-Thrust Optimization</td>
<td>Medium</td>
</tr>
<tr>
<td>Mystic</td>
<td>Optimization of trajectory or entire mission</td>
<td>High</td>
</tr>
<tr>
<td>Copernicus</td>
<td>Generalized spacecraft trajectory design and optimization system</td>
<td>High</td>
</tr>
<tr>
<td>OTIS</td>
<td>Optimal Trajectories by Implicit Simulation</td>
<td>High</td>
</tr>
<tr>
<td>SNAP</td>
<td>Spacecraft N-body Analysis Program</td>
<td>High</td>
</tr>
<tr>
<td>CHEBYTOP</td>
<td>Chebyshev (Polynomial) Trajectory Optimization Program</td>
<td>Low</td>
</tr>
<tr>
<td>VARITOP</td>
<td>Variationally Calculus Trajectory Optimization Program</td>
<td>Medium</td>
</tr>
<tr>
<td>SEPTOP</td>
<td>VARITOP based Solar Electric Propulsion Trajectory Optimization Program</td>
<td>Medium</td>
</tr>
<tr>
<td>NEWSEP</td>
<td>(new) VARITOP based Solar Electric Propulsion Trajectory Optimization Program</td>
<td>Medium</td>
</tr>
<tr>
<td>Sail</td>
<td>VARITOP customized for solar sails</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 2. Availability of JPL Low-Thrust Trajectory Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MALTO</td>
<td>Freely available to NASA contractors, civil service, and academia. Commercial licenses available for a fee.</td>
</tr>
<tr>
<td>Mystic</td>
<td>NASA employees only</td>
</tr>
<tr>
<td>Copernicus</td>
<td>NASA center, government contractors, and universities with contractual affiliations with NASA.</td>
</tr>
<tr>
<td>OTIS</td>
<td>Anybody in government, academia, and industry. Subject to export control regulations.</td>
</tr>
<tr>
<td>SNAP</td>
<td>Anybody in government, academia, and industry. Subject to export control regulations.</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHEBYTOP</td>
<td>General public</td>
</tr>
<tr>
<td>VARITOP</td>
<td>NASA employees only, and universities with contractual affiliations with NASA.</td>
</tr>
<tr>
<td>SEPTOP</td>
<td>NASA employees only, and universities with contractual affiliations with NASA.</td>
</tr>
<tr>
<td>NEWSEP</td>
<td>NASA employees only, and universities with contractual affiliations with NASA.</td>
</tr>
<tr>
<td>Sail</td>
<td>NASA employees only, and universities with contractual affiliations with NASA.</td>
</tr>
</tbody>
</table>

Almost all the programs are exclusively available to NASA or government employees. Copernicus and MALTO are available to Academia, but only with contractual obligations to NASA; MALTO is also commercially available, but comes with a hefty large fee. The only tool that is accessible to anyone the public is CHEBYTOP. CHEBYTOP is an Excel/fortran tool for low-fidelity rapid analysis of low-thrust trajectories, making it similar to the Low-Thrust STOpS developed in this work. Unfortunately, CHEBYTOP was developed in the late 1960’s and early 1970’s and is no longer user friendly. Low-Thrust STOpS has similar fidelity and versatility to CHEBYTOP, but also boasts an intuitive user-friendly MATLAB script for customization. The shallow learning curve and functionality associated with STOpS makes it a viable alternative to CHEBYTOPs that can be used by academia, industry, and individuals interested in trajectory optimization.

Low-thrust optimization is inherently more difficult than high-thrust optimization because the spacecraft will be thrusting for at least 50% of the trajectory during low-thrust applications as opposed to less than 1% for high-thrust applications. When thrust is applied continuously, the magnitude and
direction of the thrust also needs to be defined continuously along the entire orbit rather than just at the beginning and the end. For this reason, the variable set that needs to be optimized becomes much larger for low-thrust applications. In order to simplify the low-thrust problem and reduce the number of variables to a reasonable amount, this work implemented the techniques presented in “Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm” [5] along with the methods presented in “Spacecraft Trajectory Optimization” [2]. While both sources were able to successfully optimize a low-thrust orbit transfer using an evolutionary algorithm, this work goes a step further by integrating the same techniques into a user-friendly suite. The suite gives the user access to the same functionality and techniques that are presented in the papers, but also allows the user to quickly and easily change the optimization parameters and the target trajectory. Additionally, each of the two sources only uses one variable simplification method and one type of evolutionary algorithm; the model island paradigm used in this work enables the use of either variable simplification method and any or all of three evolutionary algorithms. This work benefits the community by providing a tool which allows users to easily and effectively use the techniques presented in “Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm” [5] and “Spacecraft Trajectory Optimization” [2] to optimize a variety of low-thrust interplanetary trajectories. Both methods were implemented successfully and have certain advantages and disadvantages.

1.4 Structure of Paper
This paper begins with an introduction on orbital mechanics in Section 2. This section also covers the fundamental difference between a high-thrust orbit transfer and a low-thrust orbit transfer, thus allowing for a wide range of readers to understand the content, so even readers familiar with orbital dynamics are encouraged to read it. Section 3 addresses the fundamental differences between high-thrust and low-thrust trajectory optimization. In addition to the differences between the two problems, this section talks covers about the in-depth setup used in this work including the assumptions made, the composition of the variable strings, the equations of motion, and the coordinate systems—and assumptions that were made. Section 4 discusses optimization in general. After provides an overview of optimization this section begins a, as well as a detailed summary of all the algorithms used in this work. Even readers with previous knowledge in each of the three evolutionary algorithms used in this work are advised to read this section because ideas presented here may differ from the accepted standard. Section 5 presents information on how the Island Model Paradigm functions and how it was applied in this work. Section 6 illustrates the verification of the low-thrust optimization suite for two test cases. The algorithms were already verified for the completion of during Fitzgerald’s thesis and they were only slightly modified for this work, so they are were not individually verified again. Section 7 is the results. Here includes the results, another an additional test case is defined along with the lessons learned during the creation and use of the suite. Finally, Section 8 includes the conclusion and future works.
Any readers who are interested in a detailed user guide for Low-Thrust STOpS should consult the Appendix A.

1.5 Acronyms

ACO – Ant Colony Optimization
DU – Distance Units (1.496E^8 km)
EP – Electric Propulsion
GA – Genetic Algorithm
GUI – Graphical User Interface
PSO – Particle Swarm Optimization
SOI – Sphere of Influence
STOpS – Spacecraft Trajectory Optimization Suite
TU – Time Units (58.13 days)
ΔV – Change in Velocity (Delta V)
CHAPTER 2. ORBITAL MECHANICS

Some people may think that when an object is in orbit around a celestial body, such as the Earth, there are no forces acting on it. However, in reality, there are many forces acting on a satellite at any given moment. The largest force acting on satellites is gravity, but there are also several smaller forces that are not considered in this work including solar radiation pressure, drag, and the non-circular Earth effect. It is valid to ignore these smaller forces because they are orders of magnitude smaller than gravity.

Without gravity, maintaining an orbit would not be possible. A satellite in orbit has essentially achieved the perfect balance in velocity because it is going fast enough to escape gravity and avoid plummeting into the celestial body, but not going fast enough to escape gravity completely and fly away from the celestial body. Satellites farther away from the central body need less velocity to maintain their orbit. A body in orbit must either speed up or slow down to alter its orbit.

Performing a speed change requires the spacecraft to turn on its engine and point itself in the correct direction. When the spacecraft turns on its engine to adjust its velocity it is called a burn. For this work, engines will be separated into two categories: high-thrust and low-thrust.

2.1 High-Thrust Versus Low-Thrust Engines

Traditionally, spacecraft use engines that produce a relatively high amount of thrust. These types of engines are typically chemical in nature, combining a fuel and an oxidizer with a spark to create a fiery high-speed ejection of mass out of the nozzle to create thrust. This work will refer to trajectories that use these...
types of engines as high-thrust or impulse-thrust. High-thrust engines allow a spacecraft to speed up or slow down very quickly; however, they are inefficient and therefore require a lot of fuel. Due to the inefficiency of these high-thrust engines, many spacecraft developers have moved toward using electric propulsion or what will be referred to as low-thrust engines. Low-thrust engines work by exciting individual molecules in a gaseous fuel and then ejecting the molecules out of the nozzle at speeds orders of magnitude higher than traditional high-thrust engines. These engines produce a small amount of thrust because each molecule has such little mass, but since each molecule is accelerated to such a high velocity they are extremely efficient. Low-thrust engines help to dramatically reduce the mass of a spacecraft, but make the trajectory optimization much more difficult.

2.2 High-Thrust Orbit Transfer Mechanics

Consider the satellite orbiting Earth in Figure 1. The initial orbit is shown in green and labeled segment 1, the transfer orbit is in yellow and labeled segment 2, and the final orbit in red and labeled segment 3. Assuming gravity is the only force in action, a satellite’s orbit will be constant. For example, if a satellite were in the orbit 1 and neither slowed down or sped up, it would continue to orbit the central body, O, following the solid/dashed green path. Performing a burn in any direction will change the satellite’s orbit. For example, if the satellite in orbit 1 performs a burn to speed up at the location indicated by the black arrow labeled $\Delta V$, it will expand its orbit to become orbit 2. Notice that the point on the orbit 2 where the orbit expansion occurred is labeled $\Delta V$. When a spacecraft turns on its engine to accelerate in a certain direction it’s called a burn.
orbit where the spacecraft burns does not move, but rather the opposite side of the orbit is extended. Conversely, if the spacecraft were to slow down at that point, the opposite side of the orbit would shrink towards the central body. Now consider a satellite that is in orbit 2, the elliptical transfer orbit; once it reaches the point indicated by the black arrow labeled $\Delta V'$ it can burn again, increasing its speed, to enter orbit 3. Just like when the satellite entered the transfer orbit, the orbit is extended on the opposite side until it matches orbit 3. The orbit transfer just described required two burns. Both burns take some amount of time to complete, but for high-thrust engines that time accounts for less than one percent of the total orbit transfer, so it is reasonable to assume they are instantaneous. This assumption greatly simplifies the problem.

![Figure 1. High-Thrust Orbit Transfer [7]](image)

2.3 Low-Thrust Orbit Transfer Mechanics

Spacecraft that employ low-thrust engines apply less thrust over a long period of time to achieve the same change in velocity as high-thrust engines. Low-thrust spacecraft are burning for over 50% of the duration of the transfer
orbit, so it is no longer valid to assume that the burns are instantaneous. Without this assumption, the orbit transfer calculations becomes more difficult. Rather than simply entering a transfer orbit and then injecting into the final orbit, the orbit changes continuously over the transfer time until the conditions for the desired orbit are met. Figure 2 is a good example of a low-thrust orbit transfer. It shows a spacecraft transferring from Earth’s orbit to Mars’s orbit over the course of 655 days. In this case the spacecraft applied a constant thrust of 0.33 Newtons. The trajectory spirals out towards the destination orbit as the spacecraft gradually gains speed. The difficulty behind this maneuver is knowing where to direct the thruster throughout the transfer. Figure 3 shows the thruster pointing profile that accompanies the orbit transfer in Figure 2. The angle is defined counterclockwise from the tangential velocity vector of the spacecraft. Only by finding the optimal

Commented [KJ15]: Is it that the orbit transfer is more difficult or the calculation of the orbit transfer?

Commented [KJ16]: Do you want to talk about continuous versus low thrust? You say it is continuous but then also label it as low thrust? Is it really low thrust? You might want to add something that talks about where that distinction level lies.

Commented [KJ17R16]:

Commented [SS18R16]: Not sure what I would need to clarify.

Commented [KJ19]: Write a sentence here describing the what is on the x axis and the y axis and remove the titles on the plots. The reader knows nothing about the methods and the title is confusing at this point. When you have a label you don’t need a title.

Commented [SS20R19]: I think its clear what is shown in the plot, especially with the legend.
direction to point the thruster throughout the whole orbit can the spacecraft successfully insert into the desired orbit.

Figure 2. Low-Thrust Orbit Transfer

Figure 3. Thrust Pointing Angle for Low-Thrust Trajectory
CHAPTER 3. OPTIMIZATION SETUP

Each trajectory must first be represented as a string of variables before it can be optimized. Once a trajectory is simplified to a list of variables, optimization algorithms can attempt to find the value for each variable that results in the most satisfactory trajectory. As seen in Section 2, low thrust trajectories and high-thrust trajectories are fundamentally different. This means the variables they that are simplified to simplified variable sets will also be very different. Furthermore, however, even for just low-thrust problems, there are multiple ways to represent a trajectory with an array of variables for low-thrust problems. Each different method for representing a low-thrust trajectory with variables has its respective advantages and disadvantages. This work explores two different methods for representing a low-thrust trajectory. The methods used will be referred to as the segmented method and the costate method. This section will discuss the method used by Fitzgerald to describe a high-thrust trajectory, assumptions and simplifications that were made for the low-thrust problem, and the segmented and costate methods that were investigated in this work.

3.1 High-Thrust Trajectory Variable Reduction

The advantage to high-thrust trajectories is the assumption that all burns are instantaneous. That means any trajectory only requires two burns; one to insert into the transfer orbit and one to insert into the final orbit. Continuing with that logic, any high-thrust trajectory can be described with five parameters: the initial position, the magnitude and direction of the first burn, the coast time, the magnitude and direction of the second burn, and the final position. In this...
the initial position for the trajectory is just the position of the
departure planet at the initial time and the final position is the position of the
arrival planet at the end time. Lambert’s problem is a common orbital mechanics
problem where the initial position, the final position, and the transfer time are
known, but the direction and magnitude of each burn is unknown [3]. Therefore, if
a transfer time is picked arbitrarily, Lambert’s Solution can use the start and end
positions to find the velocity at the beginning and the end of the transfer. While
arbitrarily picking the end time allows Lambert’s Solution to be used, the only
unknown remaining is the time of flight the correct time of flight is still
unknown.

Fitzgerald applied this method when creating the variable arrays for high-
thrust trajectories [4]. Each variable in a specific array or solution represented the
time for a leg of the journey. For example, consider a trajectory where a
spacecraft starts at Earth, performs a gravity assist at Mars, performs a second
gavity assist at Jupiter, and then arrives at Saturn. The variable array that
describes this trajectory would look like

\[ t_0 \ t_1 \ t_2 \ t_3 \]

where \( t_0 \) is the date that the spacecraft departs Earth, \( t_1 \) is the time required to
get from Earth to Mars, \( t_2 \) is the time required to get from Mars to Jupiter, and \( t_3 \)
is the time required to get from Jupiter to Saturn. Lambert’s solution is then used
to solve for the total change in velocity (\( \Delta V \)) required by that trajectory. Some
trajectories will require a lot more \( \Delta V \) than others, but by optimizing \( t_0, t_1, t_2, \)
and \( t_3 \) the most efficient trajectory can be found. Of course, while there are other

Commented [JR22]: Do you mean in low-thrust applications? It's confusing because you are talking about high-thrust applications here.

Commented [KJ23]: So you confused me here, you said you pictured a transfer time but then said the only remaining unknown is time of flight. I'm assuming you mean because you picked it a time of flight you never know if it is a correct.

Commented [JR24]: Should this be “change in velocity”?
ways to judge the quality of a trajectory, but delta V is the most common metric used in orbital mechanics.

What is important to note about this method for describing high-thrust orbit transfers is that every possible variable array that is generated is a viable solution to the problem. For some solutions, the delta V may be completely unrealistic, but it would still be a solution that gets the spacecraft from the start position to the correct end location. Additionally, Lambert’s Solution is the only function needed to evaluate each solution, which is beneficial because it is relatively computationally inexpensive.

3.2 Low-Thrust Problem Setup

The low-thrust problem required a completely different setup than the high-thrust suite created by Fitzgerald. Certain simplifications and assumptions were made, which lead to the following arrangement. This section explains the coordinate system, the units, the differential equations, the propagation method, and the cost function that were used in this work.

3.2.1 2D Polar Coordinates

This work chose to implement a simplified 2D coordinate system to reduce the number of variables that were in play. A 2D polar coordinate system was adopted based on the work by Rauwold and Coverstone-Sarroll in their Near Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm [5]. Figure 4 shows how the coordinate system is defined.
The radius, $R$, is defined as the distance from the attracting center to the spacecraft. The angular position, $\theta$, is defined counterclockwise from the positive X axis to the radius vector. The radial and tangential velocity, $U$ and $V$, are defined locally at the spacecraft with the radial velocity being parallel to the radius vector and the tangential velocity being perpendicular to the radius vector. The thrust vector is 2D as well, so it can be represented with a magnitude and an angle. The thrust pointing angle, $\phi$, is defined clockwise from the tangential velocity vector. The 2D coordinate system allows a spacecraft's state to be described with 4 elements instead of 6 and the thrust profile to be described with 2 elements instead of 3. The assumption for 2D orbits is valid because the maximum inclination of any of the 9 planets is 7 degrees. Some sources assume circular orbits for planets, but this work did not use that assumption. Instead, real ephemeris data was used.

### 3.2.2 Canonical Units

All the units in this work were converted to canonical units. Canonical units reduce the size of large numbers so they are easier to compare at a glance, allow the
computer to carry greater precision, and in certain cases they may speed up an
algorithm [14]. In this work a distance unit (DU) was defined as one astronomical unit (AU). An AU is the average distance from the earth to the sun, which is $1.496 \times 10^8$ kilometers. The unit for speed was defined as the average speed of the Earth, which made the time unit (TU) equal to 58.13 days. For reference, with these units an object in Earth orbit would have a radius vector of approximately 1 DU and a velocity of about 1 DU/TU. With the defined values for a DU and a TU, the standard gravitational parameter of the Sun, $\mu$, becomes $1 \text{DU}^3/\text{TU}^2$.

3.2.3 Propagation and Equations of Motion

Both methods of low-thrust trajectory optimization used in this work require the initial state of the spacecraft to be propagated until the end time to evaluate the fitness of the solution. Ode45, a built in MATLAB propagation function, was used in this work for its ease of use and stability. The state vector, $X$, varies between variable simplification methods, but always has the following five elements in common

$$X = [... \ R \ \ \theta \ \ U \ \ V \ m ...]^T$$

where $R$ is radius in DU, $\theta$ is the angular position in degrees, $U$ is the radial velocity in DU/TU, $V$ is the tangential velocity in DU/TU, and $m$ is the total mass of the spacecraft in kg. The orbital equations of motion of these five elements for the 2D polar coordinate system are

$$\dot{R} = U \quad (1)$$

$$\dot{\theta} = \frac{V}{R} \cdot \frac{180}{\pi} \quad (2)$$

$$\dot{U} = \frac{V^2}{R} - \frac{\mu}{R^2} + \frac{T \sin(\phi)}{m} \quad (3)$$

$$\dot{V} = -\frac{\mu V}{R^2} + \frac{T \cos(\phi)}{m} \quad (4)$$
\[ \dot{m} = \text{constant, user defined function, or } \frac{T}{gI_{sp}} \]  

\[ \begin{align*}
\dot{U} &= U \\
\dot{\theta} &= \frac{\mu}{R^2} \\
\dot{V} &= \frac{\mu}{R^2} + \frac{T \sin(\phi)}{m} \\
\dot{R} &= \frac{\mu}{R^2} + \frac{T \cos(\phi)}{m}
\end{align*} \]

where each equation represents the time derivative of its respective element in the state vector in terms of components from the current state vector, \( T \) is the thrust in \( \frac{\text{kg} \cdot \text{DU}}{T U^2} \), \( \mu \) is the gravitational parameter of the Sun in \( \frac{\text{DU}^3}{T U^2} \), and \( \phi \) is the thruster pointing angle in degrees. **Kilograms remain in the unit for thrust because only distance and time were normalized for this work.**

### 3.2.4 Cost Function

The cost function is how the fitness of any solution is evaluated, **so thus it making it an critical part** to any optimization algorithm. The most common factors normally included in a high-thrust trajectory cost function are delta V and time of flight, although other parameters like heliocentric energy and final mass may also be included. These cost parameters work **fine well** assuming that every trajectory generated is a viable solution; unfortunately, that is not the case for the low-thrust methods used in this work. The cost function here focusses on finding a trajectory that minimizes the error between the final state of the spacecraft and
the desired end conditions. The cost function used in this work was adapted from "Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm" by Gerald A. Rauwolf and Victoria L. Coverstone-Carroll [5]. There are five components included in the cost function used in this work, all of which can be enabled or disabled by the user. The five components included in the cost function are radius, angular position, radial velocity, tangential velocity, and time. The equations used to calculate the cost for each component are as follows:

\[
J_R = \frac{(R_{\text{alg}} - R_f)^2}{tol_R^2}
\]  

(6)

\[
J_\theta = \frac{(\theta_{\text{alg}} - \theta_f)^2}{tol_\theta^2}
\]

(7)

\[
J_U = \frac{(U_{\text{alg}} - U_f)^2}{tol_U^2}
\]

(8)

\[
J_V = \frac{(V_{\text{alg}} - V_f)^2}{tol_V^2}
\]

(9)

\[
J_{tt} = \frac{-(tt_{\text{alg}} - tt_f)(tt_{\text{alg}} - tt_f)}{w_{tt}^2}
\]

(10)

\[
J = J_R + J_\theta + J_U + J_V + J_{tt}
\]

(11)
\[
J_R = \left( R_{\text{alg}} - R_f \right)^2
\]

Tangential velocity, \( tt \) is end time, \( tol \) is the tolerance, and \( w \) is weight. The subscripts on \( J \), \( tol \), and \( w \) indicate which component they are associated with. The subscript ‘alg’ indicates final values from the solution being evaluated and the subscript ‘f’ indicates values from the desired end conditions. The best solution in this case would exactly match the desired end conditions and minimize the transfer time, giving a cost near zero or possibly less than zero if the final time is less than the target time. The tolerance values represent what error is acceptable for the component being considered. For example, this work used a tolerance value of 0.01 for radius, angular position, radial velocity, and angular velocity, which means the algorithm is attempting to converge those values with 1% of the desired end conditions. The time based component differs from the others in that the time is not weighted as heavily as the other components. The weight for the time component allows the user to emphasize how important it is to minimize the time. A lower weight will increase the importance of finding a solution with a low transfer time, but may result in the end conditions not being as close to the desired values. The best value for \( w_{tt} \) was determined to be around 3.5 [5], which allowed the suite to explore the entire range of transfer times, but slightly prioritize solutions with the lowest transfer time. The user is able to enable or disable any of the five cost components and define the individual tolerances/weight. If the goal of a trajectory is to insert into a planet’s orbit, but not arrive at the planet’s location, then the user should disable the theta component of the cost function. The time component of the cost

Commented [KJ36]: How did you come up with this number? Through trial or through one of the papers?
function is not necessary for the costate variable construction method (discussed in section 3.3.2) because the equations of motion used automatically attempt to minimize the transfer time. A good cost in this work was considered to be anything under 10 and a great cost was considered to be less than 1.

3.3 Low-Thrust Trajectory Variable Reduction

As mentioned previously, a low-thrust spacecraft is typically burning for at least 50% of the time that it is performing an orbit transfer, so the trajectory is harder to simplify. The reduced variable set must somehow describe the thrust and thrust pointing angle of the spacecraft at every point in its orbit. Once the thrust and thrust pointing angle are described for the whole trajectory, the initial state can be propagated until the final time to calculate the trajectory. The downside to this is that orbit propagation is computationally expensive compared to Lambert’s Solution, so low-thrust trajectories take much longer to evaluate than high-thrust trajectories. Additionally, there is a large possibility that the thrust vectors and pointing angles for a given array will result in a trajectory that terminates nowhere near the desired end conditions. Unlike high-thrust optimization, not every variable string that is generated is a viable solution. Optimization in this case is focused on finding the thrust profile that will result in the lowest error between the desired end conditions and the actual end conditions. The following sections describe the two variable reduction methods.

3.3.1 The Segmented Method

The segmented method simplifies a low-thrust trajectory by dividing it into a finite number of segments [5]. Each segment can be described by a thrust
magnitude and a pointing angle as defined in Section 3.2. For this work, the thrust and thrust pointing angle describing a certain segment begin at the start of a segment are held constant until the next segment is reached. The length of each segment is determined by dividing the total transfer time by the number of segments desired. Dividing the trajectory into N segments means the thrust profile only needs to be described at N points instead of continuously throughout the trajectory. For example, consider a trajectory where a spacecraft starts at Earth and performs an orbit transfer to get to Mars. If 5 segments are used the whole trajectory is described with just 12 variables: 5 thrust magnitudes, 5 pointing angles, and 2 additional variables for the start and end time. The array that describes this trajectory would look like:

\[ [t_0, T_1, T_2, T_3, T_4, T_5, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, t_f] \]

where \( t_0 \) is the departure date, \( T_N \) is the thrust for the Nth segment, \( \phi_N \) is the thrust pointing angle for the Nth segment, and \( t_f \) is the transfer time. However, if the thrust is assumed to be constant throughout the trajectory, \( T_N \) represents a binary thrust switch variable. This means that if \( T_N \) is 1 the thruster is turned on and if \( T_N \) is 0 the spacecraft is coasting. For this scenario, the first thrust switch is always assumed to be 1 because the thruster must be on to begin the orbit transfer. With these assumptions, the array is simplified to

\[ [t_0, \phi_1, T_2, \phi_2, T_3, \phi_3, T_4, \phi_4, T_5, \phi_5, t_f] \]

where all the variables are the same as before, but \( T_1 \) does not need to be included. To summarize, a trajectory described with N segments would have 2N+2 variables if the thrust is variable and 2N+1 variables if the thrust is
The advantages to this method are that coasts can be included, the thrust can be variable, and the user has the ability to define how many segments to use. Using more segments creates more variables and makes it harder to find the optimal solution, but each segment is smaller so the thrust profile is more continuous.

3.3.2 The Costate Method

The costate method simplifies a low-thrust trajectory by representing the thrust pointing angle with three costate variables: \( \lambda_1 \), \( \lambda_2 \), and \( \lambda_3 \) \[2\]. Certain conditions of optimality are employed to express the thrust pointing angle as a function of the costate variables. In this case, the necessary conditions for optimality consist of a Hamiltonian and a function of terminal conditions:

\[
H = \lambda_1 \left[ -\frac{\mu - RV^2}{R^2} + \frac{T}{m} \sin(\phi) \right] + \lambda_2 \left[ -\frac{UV}{R} + \frac{T}{m} \cos(\phi) \right] + \lambda_3 U \quad (12)
\]

\[
\Phi = t_f + v_1 [U_{alg} - U_f] + v_2 \left[ V_{alg} - \sqrt{\frac{\mu}{R_f}} \right] + v_3 [R_{alg} - R_f] \quad (13)
\]

where \( H \) is the Hamiltonian, \( \Phi \) is the function of terminal conditions, \( \lambda_N \) are the time-varying costate variables, \( \mu \) is the gravitational parameter of the Sun, \( R \) is the spacecraft’s time-varying radius, \( V \) is the spacecraft’s time-varying tangential velocity, \( U \) is the spacecraft’s time-varying radial velocity, \( T \) is the constant thrust value of the spacecraft, \( m \) is the time-varying mass of the spacecraft, \( \phi \) is the time-varying thrust pointing angle, and \( v_N \) are the time-independent adjoint variables conjugate to the boundary conditions \[2\]. The subscript ‘alg’ indicates the value of the respective variable at the end time as provided by the optimization suite while the ‘f’ subscript indicates the desired end value of the
The necessary conditions for optimality also include the following equations for the costate variables [2]:

\[
\begin{align*}
\dot{\lambda}_1 &= -\lambda_3 + \frac{V \lambda_2}{R} \\
\dot{\lambda}_2 &= -2V \lambda_1 + U \lambda_2 \\
\dot{\lambda}_3 &= \frac{V^2 \lambda_1 - UV \lambda_2 - 2\mu \lambda_1}{R^2} - \frac{2\mu \lambda_1}{R^3}
\end{align*}
\]  

(14) \hspace{1cm} (15) \hspace{1cm} (16)

where each equation represents the time derivative of its respective costate element in terms of the components from the current state vector. Additionally, the thrust pointing angle, \( \phi \), can be expressed as a function of the costate variables through the Pontryagin minimum principle [2]:

\[
\begin{align*}
\cos(\phi) &= -\frac{\lambda_2}{\sqrt{\lambda_1^2 + \lambda_2^2}} \\
\sin(\phi) &= -\frac{\lambda_1}{\sqrt{\lambda_1^2 + \lambda_2^2}}
\end{align*}
\]

(17) \hspace{1cm} (18)

Lastly, since the final time is free the following transversality condition must be met:

\[
H(t_f) + \frac{\partial \Phi}{\partial t_f} = 0 \quad \text{yields} \quad T \frac{yields}{m} \sqrt{\lambda_1(t_f)^2 + \lambda_2(t_f)^2} - 1 = 0
\]

(19)

All of the necessary conditions for optimality and the transversality condition allow the control problem to be simplified to a two-point boundary value problem where \( \lambda_1, \lambda_2, \) and \( \lambda_3 \) describe the thrust pointing angle. The result is a trajectory that simplifies the trajectory to an array with only five variables:

\[ [t_0 \ \lambda_1 \ \lambda_2 \ \lambda_3 \ t_f] \]
where \( t_0 \) is the date of departure, \( \lambda_N \) are the costate variables to describe the thrust pointing angle, and \( t_f \) is the total time of flight. The costate variables are used to express the thrust pointing angle, \( \phi \), through the following equations:

\[
\cos(\phi) = -\frac{\lambda_2}{\sqrt{\lambda_1^2 + \lambda_2^2}}
\]

\[
X = [\lambda_1 \lambda_2 \lambda_3 R \theta U V m]^T
\]

The equations of motion for the costate variables are as shown in equations 14-16 while the equations of motion used for propagation of the costate variables are

\[
\dot{\lambda}_2 = -\lambda_3 + \frac{\mu V}{k}
\]

### 3.4 User Interface

When this work was started it quickly became clear that low-thrust optimization and high-thrust optimization are significantly different problems, so it made sense to make the high-thrust and low-thrust optimization suites two separate entities. While both suites use the same algorithms, the original STOpS is dedicated to the optimization of high-thrust trajectories and the work presented here is used to optimize low-thrust trajectories. The original STOpS by Timothy Fitzgerald featured an intricate user-friendly Graphical User Interface (GUI).

While the GUI did add a lot of simplicity for the user, it added complexity to the program itself. When the algorithms from the original STOpS were being modified for this work, the GUI became a hindrance, so it was scrapped. The GUI is still fully functional for the high-thrust optimization suite, but this work chose to utilize a master script instead. The master script allows the user to easily define their mission parameters and optimization settings, but also leaves the program...
relatively uncomplicated so that other persons may continue to work on this suite in the future.

Appendix A provides instructions and descriptions for all of the user inputs. The master script that is available to the public will have the default values set for everything, but users are encouraged to try different settings as well. Once all the inputs are defined, the user can run the script to launch the low-thrust optimization. A display will show up in the command window to tell the user which migration, island, and iteration is currently running. Once the optimization is done a trajectory graph and a thrust pointing angle graph will appear to display the best solution that was found.
CHAPTER 4. OPTIMIZATION

In this work, optimization will be defined as finding the solution that best satisfies the desired end conditions. A solution is represented by a string of variables that describes the departure time, arrival time, and thrust profile for an orbital trajectory. While an optimum can be the minimum or maximum of a function, this work exclusively aimed to find solutions that minimized the cost function.

In calculus, finding an optimum to a function is as simple as taking the derivative and solving to see where it equals zero. Unfortunately, for practical engineering applications the problems being optimized are much more difficult. The derivative of the function is usually unknown or poorly defined and there are often numerous variables with a large range of acceptable values [4]. Each variable added to the problem adds a dimension to the search place, dramatically increasing the amount of possible variable combinations. A problem with N variables would have an N-dimensional search space. In the majority of real-life applications N is greater than two; however, it is impossible to visualize a search space where N is greater than two. For ease of explanation the algorithms may be described for a two-dimensional problem, but they would function exactly the same when applied to a problem with more dimensions.

The purpose of this section is two explain how all the algorithms in this work function. Each algorithm was adopted from the original STOpS by Timothy Fitzgerald. While the algorithms are mostly unchanged, some modifications were made to allow them to handle the low-thrust problem. This section will give an
overview of each algorithm and touch on what was changed for this work. If the reader is interested in how the algorithm was originally created, they are encouraged to reference the original STOpS [4].

4.1 Local Versus Global Optimum

When optimizing a problem, it is difficult to determine if the best solution has been reached or if it is just a very good solution. The absolute best solution is referred to as the global optima, while all other good solutions are known as local optima. It is not uncommon for a problem’s search space to contain many local optima, but there can only be one global optimum. Obviously, finding the global optimum is always desired when optimizing a problem. While many optimization algorithms can only find the optimum that is located closest to their starting location, which is likely to be a local optimum, the evolutionary algorithms used in this work were all designed to find the global optimum. Evolutionary algorithms such as the genetic algorithm, differential evolution, and particle swarm optimization attempt to mimic nature. As a result, they are all stochastic methods, meaning they rely heavily on randomness. Randomness allows an algorithm to effectively explore the whole search space rather than just focusing on one general area that seems to have good solutions. By effectively exploring the search space an algorithm is much more likely to discover the global optimum. Consider Griewank’s function in the example below in Figure 5 which is Griewank’s function [15].

Commented [JR43]: Don’t want to say above/below
Griewank's function is a problem commonly used to test an algorithm’s ability to find the global optima. The equation for it is:

\[ f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \]

There are many local optima that have values very close to zero, but only the basin in the exact center has a value of exactly zero, making it the global optimum. If an algorithm were to explore only one area or Griewank’s Function then it would probably only find one of the many local optima, but if it randomly explores the whole search space it is likely to discover the global optimum instead. Another popular optimization test function is Ackley’s Function, which was used by Fitzgerald to test the algorithms used in this work [4].

4.2 Genetic Algorithm
One of the evolutionary algorithms used in this work is the Genetic Algorithm (GA). This work utilizes a modified version of the genetic algorithm developed by Timothy Fitzgerald in his master's thesis, STOpS. For more information on what texts were used in the creation of this algorithm the reader is encouraged to reference Fitzgerald [4]. The goal of this section is to provide an overview of how the genetic algorithm functions and to describe the different options that are available for the user to customize.

GAs strive to mimic natural selection; a theory by Charles Darwin [4] where the traits possessed by the fittest members of a population are more likely to be passed on to the next generation, eventually resulting in a population where most of the members possess desirable traits. GAs start with a randomized list of potential solutions. The entire group of solutions is called the population, and each individual solution is referred to as a member. The fitness of each member is then evaluated via the cost function. The best members of the current population then mate to create offspring which will compose the next generation of the population. Offspring can inherit certain traits from each of the two parents. Additionally, each member has the chance to undergo mutation, which causes part of a member to change randomly. This trend continues until the algorithm reaches a certain number of generations or a threshold cost is reached. The best member from the final generation is the optimized solution.

4.2.1 Binary Versus Continuous Genetic Algorithms

There are two primary groups of GA’s: binary and continuous [4]. These categories describe how each member is composed. As mentioned previously,
each member contains a series of variables that make up a solution. The variables can be binary or continuous.

Binary GAs were the first type that were developed. As mentioned before, each member in a GA is made up of variables strung together in a certain order. For a binary GA, each variable is expressed in binary. A binary variable with four bits can express a maximum number of 15, which would be expressed as 1111. On the other hand, 0 would be expressed as 0000. Stringing multiple numbers together, the variable string [5 6 9] would be written as [010101101001], where 5 is 0101, 6 is 0110, and 9 is 1001. The setback to a binary GA is that only integers can be expressed and the search space is discrete. To express larger values or floating-point number a much longer string would be required.

This work uses a continuous GA, although the numbers are converted to binary for the mating operation. In a continuous GA, the value of each variable can be any integer or floating-point value between its lower and upper bounds. This allows for the expression of much larger numbers with higher precision. The only real limit is the precision limited by the computer itself [4].

4.2.2 Selection Methods

The selection method used in a GA determines which members are allowed to mate to form the next generation. Choosing the correct selection method can significantly increase a GAs effectiveness. A good selection method will allow the algorithm to thoroughly explore the search space without prematurely converging on any solution. Systematic exploration of the search space increases the probability of discovering the global optimum; however, too
much exploration will result in excessive function evaluations. The selection methods described below are all available to the user.

The first selection method is called 'random selection.' As one might suspect, this method selects a random number of members from the current population to mate and form the next generation. This method is very effective at exploring the search space; however, with no advantage to solutions with a lower cost, this method is highly unlikely to yield any useful results. Generally, this method is discouraged.

A more popular method is 'natural selection.' This method specifically is what attempts to imitate Darwin’s theory of natural selection, hence the name [4]. For this method, the best N solutions are selected, where N is determined by the user. N must be greater than zero but less than the size of the population. It is recommended that N is at least 50% of the population size to ensure the search space is explored adequately. After the costs of the current members are evaluated, the best N members are selected to be able to mate. Mating continues between the selected members until a whole new population is created. Additionally, there is a feature called elitism, which allows the best solutions to automatically survive to the next generation, thus ensuring that the best solution will not be lost. The number of 'elite' solutions is recommended to be only 1 or 2, to ensure there are not too many identical solutions between generations. ‘Natural selection’ is very effective and is a popular choice being the original default selection method in this work the most popular method used in literature.
The selection method used primarily in this work is the 'tournament method'. This method was derived from "Genetic Algorithms, Tournament Selection, and the Effects of Noise" by Brad L. Miller and David E. Goldberg [12]. The tournament method in this work was developed from literature and was not featured in the original STOpS [4]. In this method, the population of the current generation gets divided into groups of N members. The number of members in each group, N, is determined by the user. Each group is a separate tournament. The victor from each tournament is the competitor with the lowest cost. All the victors are allowed to reproduce to form the next generation. Elitism is also an option for this method. By pairing up solutions at random, this method allows some less than optimal solutions to survive, which helps explore the search space.

Another available selection method is ‘thresholding’. Instead of picking how many solutions to select, the user sets a cost threshold. All of the current members whose cost is less than the threshold will be allowed to mate. The difficulty with this method is the user must have a reasonable guess as to what an appropriate threshold may be. If no members meet the threshold then the next generation will be completely randomized. On the other end, if every single member is below the threshold then every member can mate. Neither of these situations is desirable; the former means the algorithm is purely relying on randomness and the latter means the algorithm is no longer making progress. Even if a reasonable threshold is chosen, the algorithm can only guarantee a solution as good as the cost of the threshold. Although it was not implemented in
this work, a solution to these problems is a variable threshold method. A variable thresholding method would increase the threshold if no members met the requirement and decrease the threshold if too many members met the requirement.

The last selection method available in this work is the ‘weighted random’ method which is also referred to as the ‘roulette’ method. Similar to other methods, the user dictates the number of members that will be selected, N. Every member in the population is assigned a probability. The best solutions get dealt a higher probability. The probabilities can be generated using the cost of each member or the rank. Assigning probability based on cost results in the best members having a proportionately higher probability than the worst members. On the other hand, assigning probability based on rank, means that the probabilities are linearly distributed, so an ideal solution may not have a much higher probability than a poor solution. A rank based probability assignment may be better at exploring the search space since it gives poor solutions a higher chance to survive, but ultimately both methods are effective. Once the probabilities are assigned, a random number is then generated that determines which solution is selected. That solution is added to the mating pool and then this process repeats until N solutions have been selected. Like all other methods, elitism is an option. The advantage to this selection method is that even the members with the worst solutions have a small chance to mate. While this may seem counterintuitive, it is an excellent way to make sure the whole search space is explored.

4.2.3 Mating Methods
After the selection process the members are ready to mate. During mating two random members of the selected mating pool are chosen as parents. When the parents mate, certain sections of each member get shared between the two, forming two unique children. The two children are then added to the population of the next generation. Since the parent selection process is random, it is possible for some parents to mate more than once and others not to mate at all. That is one reason why elitism is implemented; the best solution from the current population is guaranteed to pass on its traits to the next generation, regardless of whether it mates or not. This helps prevent the loss of the best solution. The mating method differs between a binary GA and a continuous GA. Both methods are described below along with options available to the user.

For a binary GA mating is often referred to as ‘crossover.’ As mentioned previously, for a member in a binary GA the whole array of variables is represented as one large binary string. In this work, a crossover is performed by converting a continuous variable string into binary. For crossover to occur, two parents are selected along with a number of crossover points. The crossover points can be located at any locations along the binary strings, including in the middle of variables; however a random number check must be completed to decide if crossover occurs. Provided the crossover check is successful, the sections between each crossover point are swapped between the two parents creating two unique children. If no crossover occurs the children are identical to the parents. For example, consider two parent members whose continuous representations are:
\[ Parent_1 = [ 15 \ 4 \ 9 ] \]
\[ Parent_2 = [ 3 \ 12 \ 5 ] \]

Once converted to binary the members become:
\[ Parent_1 = [ 111101001001 ] \]
\[ Parent_2 = [ 001111000101 ] \]

For this example, there will be three cross points which all pass the crossover check. The crossover points are after bit 1, bit 6, and bit 9. After crossover occurs the children look like the strings shown below where bits from parent 1 are highlighted in blue and bits from parent 2 are highlighted in red.
\[ Child_1 = [ 01110100001 ] \]
\[ Child_2 = [ 10111001101 ] \]

Now if the binary children are translated back to continuous variables they become:
\[ Child_1 = [ 7 \ 4 \ 1 ] \]
\[ Child_2 = [ 11 \ 10 \ 13 ] \]

Both children are significantly different than the parents, which is because the cross points are in the middle of the individual variable strings. This allows the algorithm to explore more of the search space, but sometimes results in a variables value being outside its limits. If that is the case, the variable is rounded to the nearest limit.

In this work the user can choose ‘uniform crossover’ or ‘random crossover.’ ‘Uniform crossover’ means that every single bit has a chance for crossover while ‘random crossover’ chooses a number of crossover points
defined by the user at random locations along the binary string. The crossover example shown above would be ‘random crossover’ with three crossover points. In this work, after crossover-style mating is complete, the binary strings are converted back into continuous variable arrays.

A mating method that is more conducive to continuous GAs is the ‘blending’ method. This method does not require any members to be converted to binary. In this case, mating compares the same variable from each member and generates a similar value for both children. The equation used for mating is

\[ Child = \beta \times Parent_1 + (1 - \beta) \times Parent_2 \]  

where \( \beta \) is a randomly generated number called the blending factor. For example, consider the continuous version of the parents that were used from the crossover example. With a blending factor of 0.3 the child would be

\[ Child = [ 6.6 \ 9.6 \ 6.2 ] \]

A second child can then be created using the same two parents and a new blending factor. The downside to this method is the variables of the two children will always be within the two corresponding variables of the parents, which severely limits the algorithms ability to explore. A way to circumvent this issue is to increase the range for possible \( \beta \) values. Allowing \( \beta \) to range from \(-\delta\) to \(1 + \delta\) creates the potential for a child’s variables to extend beyond the limits of its parents. For this mating option, \( \delta \) is determined by the user.
4.2.4 Mutation

Any member in a GA, regardless of the selection method and mating method, has a chance for mutation. In this context, mutation is an event that occurs causing that causes part of a member to assume a new random value between the variable bounds. Mutation always occurs after mating. The probability for mutation was 5% for this work, but the user can change that parameter if they desire. Mutation gives the algorithm the opportunity to explore new areas of the search space and revisit areas that are no longer being looked at. While mutation does not always result in a member with a lower cost, it helps prevent a GA from converging on a solution prematurely. If mutation results in a member with a much higher cost, there is no real concern because that member won’t be selected for the next breeding population.

Mutation may be applied to each variably individually or to the variable string. In the case where variables mutate individually, the mutated variable becomes a random value within the variable limits. If mutation is going to be applied to a string, the string is first reconfigured into binary form. Then every bit in the binary variable string has the chance to undergo mutation. A mutated bit simply switches from 0 to 1 or vice versa. The string method results in additional exploration, but since any bit can flip there is a chance for variables to be mutated such that they are no longer within their bounds. For that reason, each member is corrected after this method of mutation; if any variables are higher than their respective maximum bound they are set to the maximum and variables
lower than the minimum bound are set to the minimum. This work chose to mutate each variable individually [4].

4.3 Differential Evolution

Differential evolution or DE is remarkably similar to a GA. Both methods start with an initial population, select certain members to combine, and create offspring to form the next generation [4]. The algorithm continues until a certain number of generations is reached. The difference between DE and a GA lies in the way that solutions are selected and the way the offspring are formed.

4.3.1 Mating/Mutation

Rather than members mating to form the next generation, the process in DE is called mutation; not to be confused with mutation in a GA. In this form of mutation, a mutant vector is created for every member of the current population. The mutant vector is created by selecting a member from the current population to be a base vector and then applying a difference vector. The difference vector allows every variable in the base vector to be slightly perturbed. The equation used for calculating the mutant vector is [4]:

\[
\hat{\mathbf{v}}_i = \hat{x}_{r_0} + F (\hat{x}_{r_1} - \hat{x}_{r_2})
\]  

where \(\hat{\mathbf{v}}_i\) is the mutant vector, \(\hat{x}_{r_0}\) is the base vector, \(F\) is the scale factor, and \(\hat{x}_{r_1}\) and \(\hat{x}_{r_2}\) are vectors used to create the difference vector. The scale factor is a value between 0 and 1. Next, the mutant vector and the original member come
together to form two parent vectors. The chance for crossover allows certain traits to be flipped from the mutant vector to the original and vice versa. The two resulting vectors are the trial vectors and there are twice as many as the original population. The selection process narrows down the trial vectors so that only half survive to the next generation, making each population the same size. The DE in this work allows the user to specify the scaling factor method, the survival method for the trial vectors, and the selection method for the base vector.

4.3.2 Scaling Factor Method

The scaling vector determines how much the difference vector will affect the base vector. Values are recommended to be greater than 0.4 but less than 1 to provide enough difference for effective searching [4]. The methods available for the scaling factor are ‘constant’, ‘jitter’, or ‘dither’. The ‘constant’ method uses the same user defined scaling factor for every difference vector in every generation. The remaining methods both use a range of user defined scale factor values. In the ‘jitter’ method, a randomly generated scale factor within the defined limits is generated for every difference vector in every generation. Lastly, in the ‘dither’ method, a randomly generated scale factor within the defined limits is generated for each generation; each difference vector within one generation uses the same scale factor. All three methods are effective, but the randomness of the ‘jitter’ method is best for effectively exploring the search space [4].

4.3.3 Selection of Base Vector and Difference Vector Contributors

The two vectors chosen to construct the difference vector, $\bar{x}_{r1}$ and $\bar{x}_{r2}$, are always chosen at random from the current population to ensure the DE algorithm
does not converge too early. This means each member may be used more than once or not at all. The base vector on the other hand is chosen by one of three methods: ‘random’, ‘best so far’, and ‘random best blend’. The only other requirement for these three vectors, \( \bar{x}_{r0}, \bar{x}_{r1}, \) and \( \bar{x}_{r2} \), is that each is unique. In ‘random’ base vector selection each population member is used once with no preference to which one has the lowest cost. The ‘best so far’ method uses the member with the lowest cost as the base vector for every single mutant vector. The last method, ‘random best blend’, is a combination of the two previous methods. For this method, linear interpolation is used to calculate a vector between the member with the lowest cost and a randomly selected member. To determine where on the line the base vector lies a number is generated between 0 and 1. Values close to 0 will result in the base vector being closer to the member with the lowest cost.

4.3.4 Selection of Survivors

For DE, the number of trial vectors generated is twice as large as the initial population. In order to move to the next generation, half of the trial vectors are selected to survive and the other half are eliminated. The two selection methods available here are ‘natural selection’ and ‘tournament’. The ‘natural selection method’ allows the members with the best cost to survive. The ‘tournament’ method is exactly like the tournament selection method for the GA. All of the trial vectors are assigned to tournaments with N members each, where N is determined by the user. The tournament competitor with the best cost is the winner. The trial vectors with the most wins become the survivors. The
‘tournament’ method allows less fit solutions to survive, which keeps the DE from eliminating any part of the search space too quickly. Both methods are elitist methods so there is no possibility for the best solution to be discarded.

4.4 Particle Swarm Optimization

Particle Swarm Optimization or PSO is meant to mimic a hive of bees searching for flowers [4]. In this analogy, a bee’s position is a variable string representing a solution. As the bees fly around they communicate with other bees around them to share information about the best-known flower location. Eventually all the bees end up converging on the optimal flower location.

Typically, in PSO the bees are flying back and forth from a set location, but that doesn’t add any value for this problem; in this work the bees are given a random initial position and velocity that allows them to explore the search space [4]. As time goes on there are three factors that affect a bee’s velocity: its velocity at the previous time step, the location of the best solution it has found, and the location of the best solution another bee has found. If tuned correctly, the bee’s velocity will cause it to arrive at the optimal location.

4.4.1 Particle Motion

When the PSO algorithm initializes, a population of bees is generated. Each bee is given a randomly generated initial position and velocity vector. The user defines how many iterations the bees’ states will be propagated for. The position of each bee is a variable string representing a possible solution; the variable bounds must be within the limits defined by the problem. The velocity vector for each bee contains the speed at which it is flying though each
respective dimension. The maximum velocity is defined by the user, but is scaled for each variable. For example, if the user picks a maximum speed of 0.5, a bee with maximum velocity in one particular coordinate would move halfway through the range of that respective variable in one time step. Since velocity is also randomly generated, the bees can be moving at any speed between the positive and negative maximum velocity. Eventually, a bee’s velocity will cause it to pass out of the search space. When this occurs, the bee’s position is set to the edge of the search space and its velocity flips signs. This keeps all the particles inside the solution space.

The acceleration of the bees is what enables them to converge on the optimal solution. As mentioned previously, three factors affect a bee’s acceleration: the velocity at the previous time step, its current best-known solution, and the best-known solution communicated to it from other bees. The equation used to calculate the new velocity is as follows [4]:

\[ v_{\text{new}} = c_1 v + c_2 (p - x) + c_3 (g - x) \]  

(23)

where \( v_{\text{new}} \) is the new velocity, \( v \) is the current velocity, \( p \) is that specific bee’s best-known location, \( g \) is the best-known location communicated from another bee, and \( x \) is the current position of the bee [4]. The three \( c \) terms are the bees’ confidence in each piece of information. Increasing the confidence in a bee’s currently velocity, \( c_1 \), would encourage the bee to keep moving the same
direction, which would help it explore the search space. Increasing $c_2$ or $c_3$ would encourage the bee to move towards the best-known solutions. The value of $c_1$ is defined by the user and kept constant throughout the time span, but it must be between 0 and 1, which signifies deceleration. This is important because the bee will eventually need to come to a stop to converge on a solution. The values for $c_2$ and $c_3$, however, can vary. The user enters a maximum confidence term for $c_2$ and $c_3$ called $c_{\text{max}}$. Each iteration, a number between 0 and $c_{\text{max}}$ is chosen for $c_2$ and $c_3$. While some literature found it beneficial to make $c_1$ and $c_{\text{max}}$ dependent, Fitzgerald found it more effective to give the user control over both values [4].

4.4.2 Informants

One of the most critical components of the PSO algorithm is the bees’ ability to communicate with each other. The best-known solutions that are communicated throughout the swarm will affect the velocity of every bee. If the same solution is shared on every iteration then it will dominate the velocity of every bee, causing them to converge on a solution prematurely. On the other hand, if too many solutions are shared then the search becomes extremely random. The user chooses the number of informants, $K$, that will be included for the algorithm. For the algorithm to perform adequately, $K$ must be high enough to promote exploration of the search space, but low enough to enable the particles to converge [4].

4.5 Ant Colony Optimization

Ant Colony Optimization or ACO is meant to mimic ants foraging for food. Ants can communicate with each other indirectly via stigmergy [4]. As ants
explore, they deposit a chemical called pheromone. If an ant finds a desirable path it will lay down more pheromone. When different ants come across the same path later their decisions are influenced by the traces of pheromone present. Over time the pheromone evaporates on unused paths and build up on the most popular paths. Eventually all the ants will converge on the best path.

ACO was used in the original STOpS by Timothy Fitzgerald, but was not used in this work because it was not easily applicable for the low-thrust problem. Typically, ACO is applied to a round trip problem. The classic example is the traveling salesman problem. In round trip problems ants leave from a start node, hit every required node, and then return home all whilst trying to minimize the cost. Orbit optimization, on the other hand, is a one-way problem, making it difficult to apply ACO. Fitzgerald was able to implement ACO with some success on the high-thrust problem, but with significant modification to what was typically found in literature. Unfortunately, the adaptations made by Fitzgerald to apply ACO to high-thrust optimization do not translate to low-thrust optimization. In the high-thrust STOpS the user could include multiple flybys in their trajectory. In that case each planet would become a node. In this work, there are only two nodes: the departure planet and arrival planet. Adapting ACO for use in this work was deemed to be a time-consuming venture which may or may not have yielded results. With the approval of Fitzgerald, ACO was omitted from this work.
CHAPTER 5. GENERALIZED ISLAND MODEL

The generalized island model [6] is a method that enables multiple algorithms to assist each other via solution sharing. Every algorithm has strengths and weaknesses, but sharing solutions allows algorithms to combine their respective strengths and overcome their weaknesses. Each algorithm included is referred to as an island. The number of islands used and the way they are linked with each other is called a topology. The ability to quickly change islands or topology gives the island model versatility, and enables the user to customize the optimization for the specific problem at hand.

The action of different islands sharing solutions is known as migration. Migration involves two major parts: selection and replacement. When the time comes for migration, each island must select a number of solutions to share with the other islands. The method for dictating what solutions get shared is an island’s selection policy. Additionally, an island must choose what solutions it wants to keep from the list of shared solutions. The method for dictating which solutions are kept is an island’s replacement policy. Controlling the selection and replacement policies prevents any one algorithm from being overpowered by the others, especially if the islands being used have different numbers of solutions per generation/iteration.

5.1 Topology

The topology of the island model describes how many islands there are, which type of algorithm each island is, and how they are all connected. Topology
is perhaps the most important aspect of the generalized island model. The purpose of the island model is that the user does not need to know which algorithm or parameters are best for a specific problem. Including a variety of algorithms and settings enables the island model to cover all possibilities. The islands that are most suited to the problem will most likely dominate the optimization process. The trade off with including more islands is that the program will be more computationally expensive and take longer to run. While the island model is intended to be run with a handful of islands, the user is encouraged to become familiar with each island’s performance individually before using the island model. If the user can make an educated guess as to what algorithms to include, rather than just using every possible combination, then the island model will be much more efficiently.

5.2 Migration

Migration is key to the island model’s success. The goal is to have a balanced migration policy so that all algorithms have a chance to contribute. If the migration policy isn’t chosen correctly, one island may dominate the optimization process, which defeats the whole purpose of the generalized island model. The two migration policies investigated by Fitzgerald are the synchronous policy and asynchronous policy. In the synchronous policy, all the algorithms share solutions at the same time, meaning migration can only occur at the pace of the slowest island. Alternatively, in the asynchronous policy the islands are sharing solutions with each other continuously, which means no island has to wait for another, but an island could be rendered useless if it is too
slow [4]. The migration policy used in this work is the synchronous policy; it may be slower, but it ensures all algorithms are contributing equally. For more details on the two migration policies an interested reader is encouraged to reference the original STOpS by Fitzgerald.

5.3 Selection

Each island has a selection policy to determine which solutions it will share with the other islands. The number of solutions shared will depend on the method picked; however, the number of solutions that is shared is not necessarily the same number that will be accepted by the other islands. The selection policies available to the user are: 'random', 'natural selection', 'threshold', 'rank weighted', and 'cost weighted'. For the 'random', 'natural selection', 'rank weighted', and 'cost weighted' methods the user chooses how many solutions will be selected and for the 'threshold' method the user chooses what the cost threshold is. These options are the same as the selection policies for a genetic algorithm. For more details on how each method operates refer back to section 4.2.2.

5.4 Replacement

Just as every island must have a policy to decide which solutions to share, it must have a policy to dictate which solutions to accept. The replacement policy determines which solutions an island will accept out of the ones that are shared with it. All or some or the shared solutions may be accepted, but every accepted solution is added to the initial population for that island. The options available to the user for replacement policies are: 'all', 'random all', 'best n', 'threshold'.
These methods are similar to options seen earlier in this work, but will be briefly explained to avoid confusion. The ‘all’ method accepts every shared solution while ‘random all’ accepts N random solutions from the ones that were shared, where N is a number defined by the user. For ‘best n’ the algorithm accepts N solutions with the lowest cost of those that were shared. The ‘threshold cost’ method accepts every shared solution that is under the threshold cost and similarly, the ‘threshold percent’ method accepts a certain percent of the best shared solutions, where the percent or threshold are defined by the user.
CHAPTER 6. VERIFICATION

Every algorithm used in this work was previously verified by Fitzgerald during the creation of high-thrust STOpS. Since each algorithm was only modified for this work and not created from scratch, re-verification of each algorithm was deemed unnecessary. For details on the verification process used by Fitzgerald the interested reader is encouraged to refer to his literature [4]. While each algorithm was proven previously, this work still needed to be performed adequately on low-thrust problems. Two test cases were performed to evaluate the suite’s ability to optimize a low-thrust trajectory. The tests cases involve two low-thrust trajectories with known near-optimal solutions that were adopted from “Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm” [5]; the first test case is a constant thrust, variable time orbit transfer from Earth to Mars and the second test case is a variable thrust, fixed-time planet-to-planet transfer from Earth to Mercury. Each test case was attempted with both the segmented and costate variable simplification method.

6.1 Test Case 1

The first test case used to validate the low-thrust suite was a constant thrust, variable time transfer from Earth’s orbit to Mars’ orbit [5].

6.1.1 Problem Definition

The first test case used to validate the low-thrust suite was a constant thrust, variable time transfer from Earth’s orbit to Mars’ orbit [5]. The source of the test case assumed circular orbits for Earth and Mars, so the start date was
insignificant. However, this work used real ephemeris data to generate 2D orbits for Earth and Mars and the start date was arbitrarily selected to be January 1st, 1994. The thrust and mass properties were chosen to be consistent with [5]:

\[ T = 3.787 \]
\[ m_0 = 4545.5 \]
\[ \dot{m} = -6.787e^{-5} \]

where \( T \) is the spacecraft’s thrust in Newtons, \( m_0 \) is the initial wet mass of the spacecraft in kilograms, and \( \dot{m} \) is the mass flow rate of the propellant out the nozzle in kilograms per second. The target time of the trajectory was 195 days, but was given a margin of +/- 20 days for both variable construction methods.

6.1.2 Island and Optimization Settings

The island model paradigm options and individual algorithm parameters were exactly the same for both variable construction methods. The settings are illustrated below in Table 3 - Table 6. Although each island can be configured uniquely, for this trial the selection method, the number of selected solutions, the replacement method, and number of replaced solutions were identical for every island. Additionally, since each algorithm is was only used once, the parameters are were only defined once. If an algorithm were was used for more than one island, the tunable parameters would have to be defined separately for each island that algorithm was used for.

Table 3. Island Model Parameters for Test Case 1

<table>
<thead>
<tr>
<th>Migrations</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Islands</td>
<td>GA, DE, PSO</td>
</tr>
<tr>
<td>Selection Policy</td>
<td>Natural Selection</td>
</tr>
</tbody>
</table>
6.1.3 Results
The results obtained by this work differ slightly from the results that accompany test case 1, but that is mostly due to the fact that this work used real ephemeris data while the source of the test case assumed circular orbits for Earth and Mars. When using ephemeris data, the start and end conditions for the trajectory are different at every point along an orbit; when circular orbits are assumed, the start and end conditions are the same no matter where the trajectory begins and terminates. Therefore, the boundary conditions used in this work are slightly different than the boundary conditions used in the source of the test case, but the results are still comparable.

The source of the test case [5] found that the optimal time for the trajectory was 193 days, but this work found it to be 211.92 days when using the segmented variable construction method. Ten segments were used in order to be consistent with the source. The transfer time found in this work is significantly longer than the time found by the source of the test case, but it makes sense because when looking at the trajectory plot in Figure 6, the spacecraft is transferring to the portion of Mars’ orbit that is farther from Earth’s orbit. If the departure date was 5-6 months later, the spacecraft would have had to transfer to the part of Mar’s orbit that is closer to Earth’s orbit, resulting in a shorter transfer time. The suite required 16.46 minutes for optimization and the found a solution with a final cost of 0.56. The optimal trajectory can be seen in Figure 6 and the accompanying thrust profile can be found in Appendix B.
Figure 6. Test Case 1: Valid Trajectory with Segmented Method

This trajectory very closely mimics the trajectory found in [Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm] which is shown in Figure 7[5].

Figure 7. Test Case 1: Near-Optimal Solutions [5]

The thrust vectors follow the same pattern in both trajectories; initially they point forward and outward, then in the middle there is a coasting segment, and finally
towards the end they point forward and inward. The interested reader is encouraged to reference the solution found in [5] for comparison. The final conditions of this solution are compared to the desired end conditions in Table 7.

<table>
<thead>
<tr>
<th></th>
<th>R (DU)</th>
<th>U (DU/TU)</th>
<th>V (DU/TU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired End Conditions</td>
<td>1.4764</td>
<td>-0.0723</td>
<td>0.8287</td>
</tr>
<tr>
<td>Optimized End Conditions</td>
<td>1.4740</td>
<td>-0.0701</td>
<td>0.8220</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>0.08</td>
<td>1.59</td>
<td>0.41</td>
</tr>
</tbody>
</table>

The highest error between the solution’s end conditions and the desired end conditions is less than 2%, so it is safe to say the solution found is valid. Adding additional migrations would result in an even better solution if the current results were not satisfactory. The one concern about this trajectory is that there is a coasting period in the middle, meaning the spacecraft may be taking longer than necessary to complete the trajectory. In an attempt to encourage the algorithm to find a faster trajectory, the author decreased the weight on the time portion of the cost function was decreased. The resulting solution did not take any less time, but had significantly higher errors for the end conditions, so the author chose to stick with the original solution was kept. Despite the coasting segment in the middle, the segmented method still performed well, giving a maximum error of only 1.59% for the radial velocity.

The costate method also performed admirably for test case 1. The constant thrust restriction forced the suite to find the most optimal solution where no time is wasted for coasting. The resulting trajectory took 204.13 days, and had a cost of 0.02, and ran for 6.2 minutes. The program also ran faster than Blah Blah.
Blah but keep from mentioning the specific times unless it is in reference to show one is faster than another. Just like the segmented method, the transfer time found for the costate method is more than the time found by the source of the test case [5] because of the distance between Mar’s orbit and Earth’s orbit at the end time. The trajectory plot can be seen in Figure 8 and the corresponding thrust profile is shown in Appendix B.

![Trajectory Plot](image)

**Figure 8. Test Case 1: Valid Trajectory with Costate Method**

As seen in Figure 8, the spacecraft successfully inserts itself into Mar’s orbit. The comparison of the final conditions and desired end conditions can be seen in Table 8.

**Table 8. Test Case 1: Comparison of End Conditions for Costate Method**

<table>
<thead>
<tr>
<th></th>
<th>R (DU)</th>
<th>U (DU/TU)</th>
<th>V (DU/TU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired End Conditions</td>
<td>1.4998</td>
<td>-0.0732</td>
<td>0.8158</td>
</tr>
<tr>
<td>Optimized End Conditions</td>
<td>1.4987</td>
<td>-0.0725</td>
<td>0.8156</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>0.04</td>
<td>0.49</td>
<td>0.01</td>
</tr>
</tbody>
</table>
The highest error for end conditions of this solution is only 0.49% for the radial velocity, indicating that this is an excellent solution.

Notice that the time for the trajectory using the costate method is about 6 days less than the solution found by the segmented method. Due to the nature of the segmented method the thrust profile is discontinuous, which results in some inefficiency of the trajectory. The continuity of the thrust profile and lack of coasts in the costate method gets this rid of those inefficiencies and enables the suite to shave a few days off the transfer time. Increasing the number of segments would reduce the errors for the segmented method, but make the search space bigger and increase the difficulty of finding the best solution. Another detail worth mentioning is the fuel efficiency of the trajectories. Even though the costate method found a faster trajectory, it spent more time thrusting overall. The solution for the costate method thrusted continuously for 204.13 days while the solution for the segmented method thrusted for only 190.73 days once the coasting segment was factored in. The costate method may be better at finding the fastest trajectory, but the segmented method has the potential to find more fuel-efficient solutions.

6.2 Test Case 2

The second test case used to validate the low-thrust suite was a variable thrust, constant time planet-to-planet transfer from Earth to Mercury [5].

6.2.1 Problem Definition

The second test case used to validate the low-thrust suite was a variable thrust, constant time planet-to-planet transfer from Earth to Mercury [5]. The
source of test case 2 [5] and this work both used real ephemeris data to generate 2D orbits for Earth and Mercury. The start date was defined to be May 6th, 1997 and the total transfer time was 355 days. Unlike test case 1, the thrust and mass properties varied with time. The spacecraft in test case 2 used a form of solar electric propulsion. The power available to the propulsion system was dictated by a model from the Jet Propulsion Laboratory [1]:

\[
P = \frac{P_0}{R^2} \left[ \frac{1.4279 - \frac{0.6139}{R} + \frac{0.0038}{R^2}}{1 - 0.2619R + 0.0797R^2} \right]
\]

(24)

where \( P \) is the power available to the spacecraft in Watts, \( P_0 \) is the available power at one astronomical unit in Watts (49717.5705 W), and \( R \) is the radial position of the spacecraft in astronomical units. In this case the solar array was restricted to a max power ratio of 1.35, meaning that if \( \frac{P}{P_0} \) exceeded 1.35, \( P \) was scaled down until the ratio was below the limit. Given the power supplied to the spacecraft from the solar arrays, the following equations were used to calculate thrust and mass flow rate [5]:

\[
T = \frac{2P}{g \cdot I_{sp}}
\]

(25)
\[ m = \frac{T}{g \cdot I_{sp}} \]  

\[ T = \frac{2P}{g \cdot I_{sp}} \]  

\[ \dot{m} = \frac{T}{g \cdot I_{sp}} \]  

where \( T \) is the thrust in Newtons, \( P \) is the available power in Watts, \( g \) is the gravity at sea level (9.81 \( m/s^2 \)), and \( I_{sp} \) is the thruster’s specific impulse in seconds. The initial mass and \( I_{sp} \) were specified to be:

\[ m_0 = 6818.3 \]  

\[ I_{sp} = 3000 \]

where \( m_0 \) is the initial mass is in kilograms and \( I_{sp} \) the specific impulse is in seconds.

6.2.2 Island and Optimization Settings

The island model paradigm configuration and algorithm settings for test case 2 are identical to the settings used in test case 1 except the island model was configured to perform 5 migrations instead of 4. Test case 2 was judged to be slightly more difficult than test case 1, so the additional migration was added to allow the suite more opportunity to converge on an optimal solution. For details on the optimization settings refer to test case 1.

6.2.3 Results
Since the time was fixed in this case, the goal of this test was more focused on the suite's ability to find a valid solution to a complex problem. Finding a solution in this case is more difficult than test case 1 because the trajectory is longer and the spacecraft must complete multiple heliocentric orbits before arriving at Mercury.

The source of test case 2 utilized 20 segments for the optimization of this trajectory and this work did the same. Initially, for the segmented variable construction method the suite had a difficult time converging on a valid solution. In this non-valid solution, the spacecraft would complete one heliocentric orbit and then attempt to converge on Mercury. The resulting cost was around 163.43 and it took 75.76 minutes to run. As seen in Figure 9, the trajectory fails to arrive at Mercury. The thrust profile for this trajectory can be seen in Appendix B.

![Figure 9. Test Case 2: Non-Valid Trajectory with Segmented Method](image-url)
The known solution for this case features the spacecraft completing two heliocentric orbits before arriving at Mercury instead of just one. While the algorithms used in this suite certainly have the potential to discover this global optimum, they kept getting stuck in the local optimum solution space that involved only one heliocentric orbit. The reason no algorithm could find the global optimum in this case is most likely because the chance of generating a random solution where the thrust profile results in two complete heliocentric orbits is very low for such a large variable string. If the population for each algorithm was increased, the suite would have a better chance of generating a solution in the global optimum basin, but evaluation would take much more time. The source of test case 2 pushed the algorithm towards the global optimum by specifying the required number of revolutions about the Sun. Specifying the number of revolutions was seen as a hindrance to the suite, but ultimately its addition was required for cases similar to test case 2. The functionality was added allowing the user to specify the number of complete heliocentric orbits that were required before converging on the target planet. This feature can be toggled on and off, but its recommended to be off unless the suite is having trouble converging on an appropriate solution. Once this featured was added, the suite had no trouble converging on the optimal solution for test case 2 using the segmented variable construction method. The final solution had a cost of 0.47 and took 77.55 minutes to run. The trajectory can be seen in Figure 10 with the corresponding thrust profile in Appendix B.
The trajectory in Figure 10 successfully terminates at Mercury with 2 complete heliocentric orbits. The comparison of the achieved end conditions and desired end conditions can be seen in Table 9.

Table 9. Test Case 2: Comparison of End Conditions for Segmented Method

<table>
<thead>
<tr>
<th></th>
<th>R (DU)</th>
<th>$\theta$ (°)</th>
<th>U (DU/TU)</th>
<th>V (DU/TU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired End Conditions</td>
<td>0.4607</td>
<td>984.1203</td>
<td>-0.0703</td>
<td>1.3167</td>
</tr>
<tr>
<td>Optimized End Conditions</td>
<td>0.4667</td>
<td>984.0595</td>
<td>-0.0672</td>
<td>1.3180</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>0.64</td>
<td>0.01</td>
<td>2.29</td>
<td>0.05</td>
</tr>
</tbody>
</table>

The maximum error in the end conditions is only 2.29%, so this trajectory was considered a valid solution. To account for the extra time, the spacecraft undergoes four coasting sections where the thruster is turned off. The coast sections are indicated by the open circles. The spacecraft essentially arrives at Mercury after 18 segments and coasts right alongside the planet for the last two segments. This indicates that the transfer time would ideally be reduced for a
true optimal transfer, but nevertheless the segmented method was able to find a solution once it was given a slight push towards the global optimum basin.

The costate method is typically not very useful for fixed time trajectories, but for this test case it worked fairly well. The equations governing the thrust profile for the costate method were derived under the assumption that time was to be minimized and that the spacecraft is constantly thrusting without coast periods. Therefore, restricting the time usually results in the costate variable construction method finding a less than optimal solution, especially in a case like this where coast periods are desirable. Nonetheless, for this test case the costate method was able to find an optimal solution with a cost of 9.84 after running for 29.99 minutes. The trajectory is shown in Figure 11 with the accompanying thrust profile in Appendix B.

Figure 11. Test Case 2: Valid Trajectory with Costate Method
Notice how in this trajectory the spacecraft dips inside Mercury's orbit and must make its way back out to arrive at the planet. This indicates that the trajectory is not optimal, because it is wasting fuel to overshoot the target and make its way back. The reason is that the time was fixed and the costate variable construction method does not allow for coasting periods. Ideally this test case would be satisfied by a trajectory with ample coasting periods; however, since the costate method requires constant thrusting the only valid solution is the inefficient trajectory seen in Figure 11. The comparison of the achieved end conditions and desired end conditions can be seen in Table 10.

| Table 10. Test Case 2: Comparison of End Conditions for Costate Method |
|---------------------|----|----------------|----------------|
|                     | R (DU) | θ (°) | U (DU/TU) | V (DU/TU) |
| Desired End Conditions | 0.4607  | 264.1203  | -0.0703 | 1.3167 |
| Optimized End Conditions | 0.4579  | 264.3773  | -0.0583 | 1.3454 |
| Percent Difference    | 0.31  | 0.05  | -9.37 | 1.08 |

The error on the radial velocity end condition is relatedly high at 9.37%, but this solution is still better than expected given the fact that the costate method is not designed for fixed time transfers. It is possible that the error could be reduced by increasing the number of migrations, but the better option is to just use the segmented method for fixed time transfers and save the costate method for situations where the start time and end time are also being optimized.

While the constant thrust restriction of the costate method did result in a less than optimal trajectory, it enabled the algorithms to converge on the global optimum without having the to specify the number of required heliocentric orbits like when the segmented method was used. The costate method specifies the
entire thrust profile using only three variables while the segmented method required 19 variables for the same exact problem. This dramatically increased the chance for the costate method to generate a random initial solution that was within the global basin. While the costate method was successful for test case 2, the segmented method is recommended for fixed time trajectories.
CHAPTER 7. RESULTS

The test cases in Section 6 provided useful insight about the suite’s strengths and weaknesses, but more information was desired. This section focuses on an additional test case that was created by the author. The goal was to investigate the optimization process for a problem where nothing was known about the solution, unlike the test cases where the near-optimal results were provided. After the results of the author defined test case, this section goes on to discuss trends and recommendations for the application of Low-Thrust STOpS.

7.1 Arbitrary Test Case

In addition to the two test cases performed in Section 6, the author created an arbitrary scenario to further test the low-thrust optimization suite. The goal of this trajectory was a constant thrust, variable time planet-to-planet transfer from Earth to Jupiter. The goal was to find a valid trajectory with the lowest possible transfer time. To make the problem more interesting, a relatively low thrust and high initial wet mass were used for the spacecraft:

\[ T = 1 \]
\[ m_0 = 10,000 \]

where \( T \) is the thrust in Newtons and \( m_0 \) is the spacecraft’s initial wet mass in kilograms. The mass flow rate and specific impulse were the same as the values used in test case 2:

\[ I_{sp} = 3000 \]
\[ \dot{m} = \frac{T}{g \cdot I_{sp}} \]  \hspace{1cm} (26)

where \( I_{sp} \) is the specific impulse in seconds, \( \dot{m} \) is the propellant mass flow rate in kilograms per second, \( T \) is the thrust in Newtons, and \( g \) is the gravity as sea level (9.81 m/s\(^2\)).

7.1.1 STOpS Settings

The island model settings and individual algorithm settings used for this test case were identical to the settings used for both test cases in Section 6 except for the number of migrations in the island model settings and the number of crossover points for the genetic algorithm. This problem was considered very difficult because of the large transfer time, low thrust, large initial wet mass, and large departure/arrival windows. To account for the increased difficulty of the problem, the number of migrations was increased to 6 for the segmented method and left at 4 for the costate method. The number of crossover points for the segmented method was increased to 15 since there were more segments being used. The rest of these settings worked great for both test cases in Section 6 so the author saw no reason to change them. The island model and algorithm settings are shown in Table 3 – Table 6.
Unlike the test cases, this problem had no precedent, so the approximate trajectory time was unknown. One option was to dramatically increase the departure/arrival time windows; however, making the time windows too big would make it more difficult for the suite to find the optimal times, so a different approach was used. The departure time window was selected arbitrarily to be from January 1st, 1994 to October 30th, 1994. Next, the target transfer time was selected to be a very large value, ideally giving the spacecraft ample time to arrive at Jupiter. If the transfer time was sufficiently long the segmented method would find a solution with plenty of coasting periods. The costate method often failed to converge for an excessively long transfer times. If the solution for the segmented method contained a significant amount of coasting segments, the transfer time was reduced and the suite was run again. This process was repeated until only a few coasting segments remained. After a couple of trial runs the ideal target transfer time was determined to be around 1900 days. The segmented method used target time of 1900 days with a margin of +45/-0 days. During the previous test cases the costate method proved to be proficient at minimizing the transfer time, even when the arrival time window was large. To encourage the exploration of shorter transfer times, the costate method was run with a target time of 1950 days and a margin of +0/-150 days.

7.1.2 Results

The biggest factor for the segmented method was determining how many segments to use. The large transfer time forced the number of segments to increase because too few segments would result in a very poor description of the
thrust profile, especially when multiple heliocentric orbits are required. However, if too many segments were used the algorithms could not find a valid solution in a reasonable time. After some trial and error, 25 segments was determined to be sufficient. With 25 segments and 6 migrations, the suite took 80.39 minutes to find a solution with a cost of 39.61. The cost of this solution was not as low as the trajectories found for test cases 1 and 2, but obtaining a better solution would have likely required a much longer evaluation time. The optimal trajectory required 1945 days and is shown in Figure 12 with the corresponding thrust profile in Appendix B.

![Earth-to-Jupiter Trajectory with Segmented Method](image)

*Figure 12. Earth-to-Jupiter Trajectory with Segmented Method*

Overall the solution produced by the segmented looks reasonable even though the cost is somewhat high. The thrust vectors seem a bit random at some points, but the trajectory spirals out nicely and arrives at Jupiter with only two coasting segments. It’s likely that increasing the number of migrations would help with the
randomness of some of the thrust vectors and possibly reduce the cost. The comparison of the trajectory’s end conditions and the desired end conditions can be seen in Table 11. The desired end conditions were identical to the position and velocity of Jupiter at the end time.

Table 11. Earth-to-Jupiter End Condition Comparison for Segmented Method

<table>
<thead>
<tr>
<th></th>
<th>R (DU)</th>
<th>$\theta$ (°)</th>
<th>U (DU/TU)</th>
<th>V (DU/TU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired End Conditions</td>
<td>4.9371</td>
<td>29.9506</td>
<td>0.0053</td>
<td>0.4608</td>
</tr>
<tr>
<td>Optimized End Conditions</td>
<td>4.9305</td>
<td>29.1649</td>
<td>0.0642</td>
<td>0.4413</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>0.07</td>
<td>1.33</td>
<td>84.68</td>
<td>2.16</td>
</tr>
</tbody>
</table>

While the end conditions for radius, angular displacement, and tangential velocity look great, the end conditions for radial velocity are off by a whole order of magnitude, 84.68%. It’s not surprising that this method had trouble converging on an optimal solution; the longer the variable string, the more difficult the problem is to optimize.

The costate method had a clear advantage for this test case. The length of the trajectory had no effect on the length of the variable string used to describe the trajectory, so this problem wasn’t much different from test case 1 in Section 6. Additionally, the costate method is much more capable of handling large departure/arrival time windows, because the variable string only consists of five variables, meaning the search space is relatively small. The costate method produced an optimal solution with a cost of 2.75 in 29.59 minutes. The optimal trajectory only required 1899.5 days and can be seen in Figure 13 with the accompanying thrust profile in Appendix B.
The trajectory produced by the costate method looks excellent. The spacecraft executes a gentle spiral outward until it rendezvous with Jupiter. The comparison of the trajectory’s end conditions and the desired end conditions is shown in Table 12.

**Table 12. Earth-to-Jupiter End Condition Comparison for Costate Method**

<table>
<thead>
<tr>
<th></th>
<th>R (DU)</th>
<th>θ (°)</th>
<th>U (DU/TU)</th>
<th>V (DU/TU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired End Conditions</td>
<td>4.9409</td>
<td>33.6600</td>
<td>0.0066</td>
<td>0.4605</td>
</tr>
<tr>
<td>Optimized End Conditions</td>
<td>4.9525</td>
<td>33.4973</td>
<td>0.0061</td>
<td>0.4719</td>
</tr>
<tr>
<td>Percent Difference</td>
<td>0.12</td>
<td>0.24</td>
<td>3.93</td>
<td>1.23</td>
</tr>
</tbody>
</table>

The costate method produced great results for such a difficult problem. With a departure window of almost 11 months and an arrival window of 150 days the suite was able to find a solution with a max error of only 3.93% in the radial velocity component. This result of this test case proves that the costate variable
construction method is a great choice for difficult low-thrust trajectory optimization problems, where the goal is to minimize the transfer time.

7.2 Lessons Learned and Recommendations

After testing both variable construction methods on a number of test cases some trends became apparent. It’s clear that each variable construction method has its own strengths and weaknesses. To obtain the best results it is critical for the user to know which method is best suited for the problem.

The costate variable construction method was able to achieve the fastest transfer time and lowest cost for both test cases that featured a variable transfer time. The costate method assumes that the spacecraft is thrusting continuously for the duration of the trajectory, so it makes sense that it is more successful at finding the fastest transfer time. Additionally, the equations of motion used for the costate variables were derived under assumptions that focused on minimizing the total transfer time. These equations of motion give the suite clear bias towards solutions with the fastest time. The segmented method attempts to encourage minimal time transfers via the time component of the cost function. Unfortunately, the pressure introduced by the cost function just isn’t high enough to effectively and consistently drive the suite towards a minimal time transfer. Theoretically the user could increase the pressure for the time component of the cost function by decreasing the respective weight; however, when this work attempted to lower the weight for the time component of the cost function, the only result was a less optimal solution. Increasing the priority on time decreased the priority on converging on the desired end conditions, which is not a helpful
compromise. When the user desires a trajectory with the minimum time, the costate variable construction method is recommended.

One noteworthy aspect of the segmented method for variable time transfers is its ability to include coasting periods. While the segmented method struggles to find a solution with the absolute minimum time, it did prove that it can find efficient solutions. In test case 1, the variable time transfer from Earth’s orbit to Mars’ orbit, the segmented method found a solution that took 6 days longer than the solution found by the costate method. The solution featured one coasting segment equating to about 21 days. The coasting portion was clearly one reason the trajectory was not as fast as possible, but it did increase the efficiency of the transfer. The total thrusting time of this solution was 14 days less than the solution produced by the costate method. While the segmented method does have the potential to add appropriate coasting segments, the arrival time window needs to be adjusted to encourage a trajectory with a reasonable transfer time. If efficiency is the highest priority for the user, the segmented variable construction method should be considered.

The segmented method takes a significant performance hit as more segments are added. Test case 1 only required 10 segments, 4 migrations, and 16.46 minutes to achieve a final cost of 0.56. When 20 segments were used in test case 2, the suite required 5 migrations and 77.55 minutes to produce a similar cost of 0.47. The trend continued in the final test case, which used 25 segments; in that case the algorithm was only able to obtain a final cost of 39.61 after 80.39 minutes and 6 migrations. Increasing the number of segments
requires more migrations and much more evaluation time to get good results. If the problem being optimized is long enough to require more than 25 segments, the user should seriously consider using the costate method.

The costate method is far superior to the segmented method when it comes to computation time. In the first test case both variable construction methods underwent 4 migrations. The costate method took 6.04 minutes while the segmented method took 16.49 minutes. The costate method used less migrations than the segmented method for the other two test cases, so it makes sense that it required less time in those cases. However, even though less time and less migrations were required, the costate method was able to find a better solution than the segmented method. The costate method should be used to quickly optimize a trajectory when little is known about the solution.

The costate method does not typically perform well for fixed time transfers. The result of the costate method for test case 2 weren’t terrible with a cost of 9.84, but the trajectory looked sub-optimal because of the way it dipped inside Mercury’s orbit before bouncing back out. During the creation of this work the costate method was tested on fixed time transfers a handful of other times and was almost never able to converge on an optimal solution. Fixed time transfers are rarely desired for orbit optimization, but if that’s what the user wants the segmented method should be used.
CHAPTER 8. CONCLUSIONS

This work successfully added the capability to optimize low-thrust interplanetary trajectories to the Spacecraft Trajectory Optimization Suite, which was originally created by Timothy Fitzgerald. The three stochastic algorithms used here are the Genetic Algorithm, Differential Evolution, and Particle Swarm Optimization. Each algorithm was successfully verified by Fitzgerald, but modified for this work. Two variable simplification methods were included in the low-thrust optimization suite, each with their own benefits. The low-thrust optimization suite created in this work and the high-thrust optimization suite created by Fitzgerald will operate as two separate programs because the input parameters are significantly different.

The low-thrust version of STOpS was verified with two known solutions presented in “Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm” by Gerald AA. Rauwolf and Victoria L. Coverstone-Carroll [5]. The first test case involved a constant thrust, time varying orbit transfer from an Earth orbit to a Mars orbit. The second test case involved a variable thrust, constant time transfer from Earth to Mercury. The solutions found in this work were found to be within an acceptable range of the known solutions and thus validate not exactly the same due to slightly different assumptions, but were close enough to confirm the validity of this work. The program was able to obtain a reasonable solution for each test case within an acceptable time; however, by increasing the number of migrations and algorithms being used it is likely an even better solution would be found given more time. The low-thrust STOpS took...
a long time to develop, but it should prove to be a very valuable tool for low-thrust deep space trajectory analysis.

8.1 Future Work

This work serves as a powerful mission analysis tool as it is, but there are aspects that could be improved upon in the future. Some improvements involve modifying the current algorithms, while others involve completely new features. The future works described here are for this work only, not the original STOpS by Fitzgerald, although some improvements are common between both works. For future works on the high-thrust STOpS the reader should reference Fitzgerald's paper [4].

The GA, DE, and PSO algorithms were successfully implemented in this work, but the local search algorithm used in the high-thrust stops was left out due to time constraints. Adding a local search algorithm may enable the program to consistently converge on even better solutions. In addition to algorithms featured in the high-thrust STOpS, persons who desire to add to this work in the future should consider adding completely new algorithms. Additional variable construction methods may also help improve the work. The generalized island model makes it easy to introduce new algorithms to the program; however, any new algorithm that is added would have to be tested independently. Persons attempting to add new algorithms in the future should consider referencing "List of Metaphor-based Metaheuristics" [11].

The 2D assumption use in this work was deemed valid considering the low inclination of all the planets; however, implementing 3D equations of motion
would make this work even more valuable. The infrastructure for the 3D ephemeris of each planet is already in place, but to implement low-thrust 3D trajectory optimization an interested person would have to create a new coordinate system that allows for the specification of the 3D state vector and the direction of the thrust vector. Along with the new coordinate system would come new equations of motion and the need to represent each trajectory as a string of variables. Neither of the variable construction methods used in this work are conductive 3D trajectory optimization without some major modifications. While the upgrade to 3D optimization would be valuable, it is probably the most difficult objective of the future works listed.

In this work gravity and the spacecraft thrust were the only forces acting on the spacecraft. While the effects of atmospheric drag and oblation on interplanetary spacecraft will be negligible, solar radiation pressure could have serious effects. The introduction of solar radiation pressure and third body perturbations would add value to this work; however, adding these perturbations may not be possible until a 3D coordinate system is implemented.

Possibly the most practical area of future work involves expanding the target orbit capabilities. Currently the suite only optimizes the transfer of a spacecraft from one planet’s sphere of influence to a second planet’s sphere of influence or orbit. One addition would be the introduction of parking orbits. The terminal conditions of the current trajectories are simply the end location and velocity of the target planet/orbit. To go one step further and inject into an orbit around a planet requires even more delta V, but is more realistic; after all what
good is the benefit of arriving at another planet if the spacecraft does not have the capability to insert itself into an orbit. Obviously, all orbit insertion would take place within a planet’s sphere of influence. Another addition would be the capability to transfer to other specific heliocentric orbits that do not involve one of the eight planets. This addition would be relatively simple because the only thing that would need to change is the end conditions.

Lastly, incorporating the option for gravity assists would be a worthy addition. For gravity assists each planet would be a node on the trajectory with certain required conditions similar to the structure used in [9]. Adding these options would significantly increase this program’s usefulness.

All the future work mentioned in this section is highly encouraged. The algorithms and code featured in this program were commented thoroughly to make the code friendly to future authors. The GUI featured in the original STOpS was intentionally omitted in this work to make future work easier. Additionally, any person attempting to modify this work is encouraged to reach out to Fitzgerald or this current author for assistance.
BIBLIOGRAPHY


Specialist Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences.


APPENDICES

A. Users Guide

The master script used to run Low-Thrust STOpS is relatively straightforward, but the user is still encouraged to educate themselves on the options available. The Island Model settings, individual algorithm settings, trajectory definition, thrust settings, and cost function settings can all be defined in the ‘STOpsS_Low_Thrust_Main_Script’ script. The script itself contains extensive commentary to illustrate the choices available, but the user’s guide in the section is more detailed.

A-1 Departure Planet and Arrival Planet

The first section of the master script that requires user input is the ‘Departure Planet and Arrival Planet’ section. As one might suspect this is where the user enters the planets that define the starting point and target end point for the trajectory. These will be referred to as the departure planet and arrival planet respectfully. This work chose to restrict the user to two planets (no gravity assists) for a couple reasons. First, the large delta V required at the periapsis of a flyby planet typically requires a high-thrust engine. Second, low-thrust trajectory optimization is computationally expensive, so the addition of multiple gravity assists would require a large amount of the time for development.

The area where the user enters the desired planets for the mission is indicated by the red text shown in Figure 14. Each body is a string corresponding to one of the 9 planets. Options are: ‘Mercury’, ‘Venus’, ‘Earth’, ‘Mars’, ‘Jupiter’, ‘Saturn’, ‘Uranus’, and ‘Neptune’. The position and velocity of the departure body
at the start time will be used to create the initial conditions for the trajectory. The position and velocity of the arrival body at the end time will be used to create the end conditions for the trajectory. If the desired trajectory is not a planet-to-planet transfer, but rather an orbit transfer, then the target body's position and velocity are ignored. In that case the spacecraft would attempt to match the conditions at a random place on the target body's orbit. All entries are case sensitive and each must be unique (i.e. the origin body and the destination body cannot be the same).

```matlab
departure_body = 'Earth';
arrival_body = 'Mars';
bodies = [departure_body, arrival_body];
```

*Figure 14. Departure Planet and Arrival Planet*

A-2 Island Model Paradigm Options

This section allows the user to input the desired parameters for the Island Model Paradigm. The required inputs are shown in Figure 15. The inputs are defined in Table 13 below:

*Table 13. Island Model Settings Guide*

<table>
<thead>
<tr>
<th>Nmig</th>
<th>The number of migrations desired. The input must be an integer greater than or equal to zero. Choosing zero would result in each island running once, but not sharing solutions. Choosing two would result in each island running three times and sharing solutions after the first and second runs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>isl_list</td>
<td>A list of all the desired algorithms in order. Options are: 'GA' (Genetic Algorithm), 'PSO' (Particle Swarm Optimization), and 'DE' (Differential Evolution). The entries must be a column vector as shown in Figure 15 and spelling is case sensitive. The order defined here is important because other variables are specific to each island. Island 1 the first entry in this vector, Island 2 is next, and so on. Any island can be used for multiple different islands, but its recommended that the tunable parameters are different for each island of a repeated algorithm.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nisl</td>
<td>The number of islands. This is a positive integer and is calculated based on what is input for 'isl_list'. The user should not modify this value.</td>
</tr>
<tr>
<td>isl_conn</td>
<td>The island connection matrix. This binary matrix dictates which islands will share solutions and which islands the solutions will be shared with. The rows represent the givers and the columns are the receivers. Example:</td>
</tr>
<tr>
<td></td>
<td>Receiver</td>
</tr>
<tr>
<td></td>
<td>Giver</td>
</tr>
<tr>
<td></td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1</td>
</tr>
<tr>
<td></td>
<td>1 0 0</td>
</tr>
<tr>
<td></td>
<td>In this example Island 1 shares with only Island 3, Island 2 shares with all 3 islands, and Island 3 shares with Island 1 and itself. This matrix is an nxn matrix where n is the number of islands being used. It is recommended that all islands share with each other and themselves. The example above would be entered as '[[0 0 1; 1 1 1; 1 0 0]]'.</td>
</tr>
<tr>
<td>rep_pol</td>
<td>Replacement policy for each island. Even though solutions are shared with an island, it doesn't necessarily accept any or all of them. This policy dictates how an island chooses which shared solutions to utilize. Options are 'random_all', 'all', 'best_n', 'threshold_cost', and 'threshold_percent'. These options are defined in Section 5 of this work. There must be one policy per island and entries are entered as a column vector. Options are spelling and case sensitive.</td>
</tr>
<tr>
<td>rep_opt</td>
<td>Replacement options for each island. If the replacement policy is 'random_all' or 'best_n' this is the number of solutions that will be accepted. If the replacement policy is 'threshold_cost' this is the threshold cost for accepted solutions. If the replacement policy is 'threshold_percent' this is the threshold percent on a 0-1 scale. If the replacement policy is 'all' this variable is not used. There must be one entry for each island and the format is a row vector.</td>
</tr>
<tr>
<td>sel_pol</td>
<td>Selection policy for each island. After an island runs once it must choose which solutions to share. This policy dictates how an island selects which solutions to share. Options are 'random', 'natural_selection', 'threshold', 'rank_weighted', and 'cost_weighted'. These options are defined in Section 5 of this work. There must be one policy per island and entries are entered as a column vector. Spelling is case sensitive.</td>
</tr>
<tr>
<td>sel_opt</td>
<td>Selection options for each island. If the selection policy is 'random', 'natural_selection', 'rank_weighted', or 'cost_weighted' this is the number of solutions to select. If the selection policy is 'threshold' this is the cost threshold. There must be one entry for each island and the format is a row vector.</td>
</tr>
</tbody>
</table>
A-3 Genetic Algorithm Parameters

In this section, the user defines the parameters for the Genetic Algorithm. The options chosen here will affect how well the algorithm performs. If the Genetic Algorithm is being used for more than 1 island the user must copy and paste the lines of code shown in Figure 16 that create the ‘GA_options(1,1)’ structure. The copied structure will serve as the settings for the Nth island that uses the Genetic Algorithm. If N islands of Genetic Algorithm are being used the structure will need to be pasted N-1 times. For example: the structure that defines the settings of the 2nd island using the Genetic Algorithm would be named ‘GA_options(1,2)’. The parameters required for each island are shown in Figure 16 and explained below in Table 14:

Table 14. Genetic Algorithm Settings Guide

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>Number of members in the population. Each member is one variable string that represents a solution. The input is a positive integer.</td>
</tr>
<tr>
<td>Ngen</td>
<td>Number of generations that will be evaluated. The input is a positive integer.</td>
</tr>
<tr>
<td>pc</td>
<td>Percent probability for a crossover to occur. Input is a number from 0 to 1.</td>
</tr>
<tr>
<td>pm</td>
<td>Percent probability for mutation to occur. Input is a number from 0 to 1.</td>
</tr>
<tr>
<td>gen_method</td>
<td>The generation method. This is how the algorithm will select which members move to the next generation. Options are 'total_random_replacement', 'tournament', 'natural_selection', 'thresholding', and 'weighted_random'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
<tr>
<td>N_keep</td>
<td>Number of solutions to keep between generations if the generation method is 'natural_selection', 'thresholding', or 'weighted_random'.</td>
</tr>
<tr>
<td>T</td>
<td>Number of members to participate in each tournament if the generation method is 'tournament'. Must be 2 or greater.</td>
</tr>
<tr>
<td>elite</td>
<td>Number of elite solutions that automatically survive to become part of the next generation. This is to prevent the best solution from being lost so its recommended to be at least 1.</td>
</tr>
<tr>
<td>threshold</td>
<td>The cost threshold if the generation method is 'thresholding'. Otherwise it is unused.</td>
</tr>
<tr>
<td>weight</td>
<td>How to calculate probabilities if the generation method is 'weighted_random'. Options are 'cost' and 'rank'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
<tr>
<td>mate_method</td>
<td>The mating method for two members. Options are 'uniform_crossover', 'random_crossover', and 'blending'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
<tr>
<td>cross_points</td>
<td>How many points are permitted between a member and its mate if the mate method is 'random_crossover'. Otherwise it is unused. Input is an integer.</td>
</tr>
<tr>
<td>OB</td>
<td>Out of bounds limit for blending. Input is a number from 0 to 1. Its recommended to be less than 0.2. See section 4 for more details.</td>
</tr>
</tbody>
</table>
A-4 Differential Evolution Parameters

In this section, the user defines the parameters for the Differential Evolution Algorithm. The options chosen here will affect how well the algorithm performs. If Differential Evolution is being used for more than 1 island the user must copy and paste the lines of code shown in Figure 17 that create the 'DE_options(1,1)' structure. The copied structure will serve as the settings for the Nth island that uses Differential Evolution. If N islands of Differential Evolution are being used the structure will need to be pasted N-1 times. For example: the structure that defines the settings of the 2nd island using Differential Evolution would be named 'DE_options(1,2)'. The parameters required for each island are shown in Figure 17 and explained below in Table 15:

**Table 15. Differential Evolution Settings Guide**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>Number of members in the population. Each member is one variable string that represents a solution. The input is a positive integer.</td>
</tr>
<tr>
<td>Ngen</td>
<td>Number of generations that will be evaluated. The input is a positive integer.</td>
</tr>
<tr>
<td>pc</td>
<td>Percent probability for a crossover to occur. Input is a number from 0 to 1.</td>
</tr>
</tbody>
</table>

```matlab
GA_options(1,1) = struct(...
    'npop',7,...
    'ngen',10,...
    'pc',0.8,...
    'ps',0.005,...
    'gen_method','tournament'...
    'n_keep',50,...
    'T',3,...
    'w/1e4',1,...
    'threshold',1000,...
    'weight','const'...
    'mate_method','random_crossover'...
    'crosso_points',20,...
    'de',0.15;
```
<table>
<thead>
<tr>
<th>sel_method</th>
<th>Selection method for the base vector. Options are 'random', 'best_so_far', and 'random_best_blend'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_method</td>
<td>Method for applying the scale factor. Options are 'constant', 'jitter', and 'dither'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
<tr>
<td>F</td>
<td>Scaling factor when applying difference vector to base vector. If F method is 'constant' then this is a (1,1) number from 0-1. For F methods of 'jitter' and 'dither' the F value is a range so the input is a (1,2) number array with the first input being the low end of the range and the second input being the high end of the range. These values are also 0-1. It is recommended in all cases that F is at least 0.4.</td>
</tr>
<tr>
<td>surv_method</td>
<td>Method for choosing survivors. Options are 'natural_selection', 'weighted_random', and 'tournament'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
<tr>
<td>T</td>
<td>The number of competitors in each tournament if the survivor method is 'tournament'. Input is an integer greater than or equal to 2.</td>
</tr>
<tr>
<td>weight</td>
<td>How to calculate selection probabilities. Options are: 'cost' and 'rank'. These options are defined in Section 4 of this work. All options are spelling and case sensitive.</td>
</tr>
</tbody>
</table>

```matlab
dE_options = struct(...
    'npop',100,...
    'ngen',30,...
    'pc',0.8,...
    'sel_method','random'...
    'F_method','jitter'...
    'F',[0.5 0.9]...
    'surv_method', 'tournament'...
    'T',2,...
    'weight','cost');
```

**Figure 17. Differential Evolution Parameters**

A-5 Particle Swarm Optimization Parameters

In this section, the user defines the parameters for the Particle Swarm Optimization Algorithm. The options chosen here will affect how well the algorithm performs. If Particle Swarm Optimization is being used for more than 1
island the user must copy and paste the lines of code shown in Figure 18 that create the ‘PSO_options(1,1)’ structure. The copied structure will serve as the settings for the Nth island that uses Particle Swarm Optimization. If N islands of Particle Swarm Optimization are being used the structure will need to be pasted N-1 times. For example: the structure that defines the settings of the 2nd island using Particle Swarm Optimization would be named ‘PSO_options(1,2)’. The parameters required are shown in Figure 18 and explained below in Table 16:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Npop</td>
<td>Number of bees (members) in the population. Each member is one variable string that represents a solution. The input is a positive integer.</td>
</tr>
<tr>
<td>vmax</td>
<td>The max velocity for each variable. This number gets multiplied by the max/min range for each variable to create a max speed specific to each variable. The velocity is on a scale from 0-1 but it is recommended to be at least 0.5 so the particles start out moving at least halfway across the variable space.</td>
</tr>
<tr>
<td>Ispan</td>
<td>How many time iterations are evaluated. This is similar to the number of generations for the Genetic Algorithm and Differential Evolution. Input is a positive integer.</td>
</tr>
<tr>
<td>K</td>
<td>Number of informants. This is the number of bees that share their best solution with all the other bees. The best K solutions are chosen from the whole population. Input is an integer greater than 0 and less than Npop.</td>
</tr>
<tr>
<td>c1</td>
<td>The bees’ confidence in their own velocity. A higher number will encourage the bees to explore the search space. This variable is on a 0-1 scale. Note that a bee’s velocity will only decrease over time.</td>
</tr>
<tr>
<td>cmax</td>
<td>The bees’ confidence in other best solutions from the informants. If this is too high it may cause premature convergence. This variable is also on a 0-1 scale.</td>
</tr>
</tbody>
</table>
Figure 18. Particle Swarm Optimization Parameters

A-6 Cost Function Selection and Parameters

The chosen cost function dictates how the trajectory is described. A variable string describes each possible trajectory. In this work, there are two main structures for the variable strings: the segmented method and the costate method. Both methods are described in detail in Section 3. The two cost function handles available are 'EP_cost_fun_segmented_2D' and 'EP_cost_fun_costate_2D'. The cost function is spelling and case sensitive. The location where this function handle is entered can be seen in Figure 19.

```matlab
PSO_options = struct(...
    'Npop', 10,...
    'vmax', 0.6,...
    'tspan', 100,...
    'k', 5,...
    'c1', 0.8,...
    'cmax', 0.6);
```

Figure 19. Cost Function Selection

Both cost functions have options that are customizable to the user. This is also where the user will decide if the trajectory is a planet-to-planet transfer or just an orbit transfer. The goal of the cost function is to have the final conditions for a member be as close to the desired final conditions as possible. The final conditions considered are radius (R), angular displacement (Theta), radial velocity (U), tangential velocity (V), and time (tt). The parameters are shown in Figure 20 and described below in Table 17:
<table>
<thead>
<tr>
<th>tolR</th>
<th>Radius convergence factor from 0-1. A value of 0.01 is recommended because that means the cost is driving to have the radius converge within 1% of the desired value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolTheta</td>
<td>Angular position convergence factor from 0-1. A value of 0.1 is recommended because that means the cost is driving to have the angular position converge within 10% of the desired value.</td>
</tr>
<tr>
<td>tolU</td>
<td>Radial velocity convergence factor from 0-1. A value of 0.01 is recommended because that means the cost is driving to have the radial velocity converge within 1% of the desired value.</td>
</tr>
<tr>
<td>tolV</td>
<td>Tangential velocity convergence factor from 0-1. A value of 0.01 is recommended because that means the cost is driving to have the tangential velocity converge within 1% of the desired value.</td>
</tr>
<tr>
<td>R</td>
<td>Radial cost switch. Choose 1 to include radius convergence in the cost calculation. Input is binary.</td>
</tr>
<tr>
<td>Theta</td>
<td>Angular position cost switch. Choose 1 to include angular position convergence in the cost calculation. Choosing 1 will result in the trajectory terminating at the arrival planet while choosing 0 will result in the trajectory terminating anywhere on the orbit of the arrival planet. Input is binary.</td>
</tr>
<tr>
<td>U</td>
<td>Radial velocity cost switch. Choose 1 to include radial velocity convergence in the cost calculation. Input is binary.</td>
</tr>
<tr>
<td>V</td>
<td>Tangential velocity cost switch. Choose 1 to include tangential velocity convergence in the cost calculation. Input is binary.</td>
</tr>
<tr>
<td>tt</td>
<td>End time cost switch. Choose 1 to include end time convergence in the cost calculation. Input is binary.</td>
</tr>
<tr>
<td>w1</td>
<td>Weight for end time convergence. Choosing a bigger number allows the end time to vary more. Choosing a smaller value puts more pressure on minimizing the time, but risks sacrificing the convergence on the desired position and velocity. The default value is 3.5.</td>
</tr>
</tbody>
</table>
A-7 Low-Thrust Mission Parameters

This is where mission parameters for the low-thrust spacecraft will be defined. Notice that there are two sections, one for each cost function. Only the section for the active cost function needs to be filled out. Some parameters are common between both the segmented method and the costate method and some are unique to the segmented method. The parameters can be seen in Figure 21 and are defined as follows in Table 18:

Table 18. Low-Thrust Mission Parameters Guide

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nseg</td>
<td>Number of segments that the trajectory is divided into for the segmented method. This is the recommended starting value is ten. This is only valid for the segmented method. Input is a positive integer.</td>
</tr>
<tr>
<td>tt_end</td>
<td>Target end time. This is approximately the desired end time for the trajectory in days although the actual end time may be higher or lower.</td>
</tr>
<tr>
<td>time</td>
<td>Upper and lower margin on target end time. The first value is the lower bound in days less than tt_end and the second value is the upper bound in days more than tt_end.</td>
</tr>
<tr>
<td>orbit_check</td>
<td>Whether or not the number of heliocentric revolutions before converging on the target is controlled. Entering 'on' allows the user to specify a number of heliocentric orbits for the spacecraft to complete before attempting to converge on the destination. Entering 'off' means the suite will attempt to find the best number of orbits on its own.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>orbits</td>
<td>The number of heliocentric orbits required before attempting to converge on the target planet/orbit if orbit_check = 'on'. Entering 1 means the spacecraft will complete 0-360 degrees of heliocentric orbit before converging. Entering 2 is 360-720 degrees of heliocentric orbit, etc.</td>
</tr>
<tr>
<td>thrust_method</td>
<td>Thrust method. Options are 'constant_thrust', 'variable_thrust', and 'equation_thrust'. The costate method is only capable of handling the 'constant_thrust' method and the 'equation_thrust' method. All inputs are spelling and case sensitive.</td>
</tr>
<tr>
<td>thrust</td>
<td>Thrust for the trajectory. If thrust method is 'constant_thrust' this is a (1,1) number for the thrust value in Newtons. If the thrust method is 'variable_thrust' then this is a (1,2) array for lower and upper bounds on the thrust in Newtons respectively. If the thrust method is 'equation_thrust' this is a function handle to calculate the thrust in Newtons. Inputs to a thrust function must be radius (AU) and Isp (s) in that order. Function handle is spelling and case sensitive.</td>
</tr>
<tr>
<td>m0</td>
<td>Initial wet mass of spacecraft in kilograms.</td>
</tr>
<tr>
<td>mdot_method</td>
<td>Method for calculating mass flow rate. Options are 'constant_mdot' and 'equation_mdot'.</td>
</tr>
<tr>
<td>mdot</td>
<td>Mass flow rate for the trajectory. If the mdot method is 'constant' this is the mass flow rate in kg/s. If the mdot method is 'equation_mdot' this is the function handle to calculate the mass flow rate; inputs to a mdot equation must be thrust (N) and Isp (s) in that order. Function handle is spelling and case sensitive.</td>
</tr>
<tr>
<td>Isp</td>
<td>Isp for the spacecraft’s engine in seconds.</td>
</tr>
</tbody>
</table>
Figure 21. Low-Thrust Mission Parameters

A-8 Earliest and Latest Departure Date

Here the user is required to enter the earliest and latest departure date from the departure body. The earliest date is called window1 and the latest departure date is called window2. The inputs can be seen in Figure 22. The format for the date is [YEAR, MONTH, DAY] with each input being an integer.

```matlab
if strcmp(fname,'G0_cost_fun_segmented_2D')
    % for Segmented Method
    thrust_opt = struct(...
        'seg',15,...
        'tt_end',200,...
        'time',[10 10],...
        'orbit_check',0,...
        'orbits',1,...
        'thrust_method',1,...
        'thrust',3.75,...
        'm0',4545.1,...
        'mdot_method',1,...
        'mdot',0,...
        'isp',3000);
else if strcmp(fname,'G0_cost_fun_costate_2D')
    % for Conway Method
    thrust_opt = struct(...
        'tt_end',200,...
        'time',[10 10],...
        'thrust_method',1,...
        'thrust',3.75,...
        'm0',4545.1,...
        'mdot_method',1,...
        'mdot',0,...
        'isp',3000);
end
```

Figure 22. Earliest and Latest Departure Date

B. Thrust Profile Plots
This portion of the Appendix contains the thrust point angle plots for all of the trajectories illustrated in this work.

B-1 Test Case 1

**Figure 23. Test Case 1: Thrust Profile for Segmented Method**

**Figure 24. Test Case 1: Thrust Profile for Costate Method**
B-2 Test Case 2

Figure 25. Test Case 2: Non-Valid Thrust Profile for Segmented Method

Figure 26. Test Case 2: Valid Thrust Profile for Segmented Method
Figure 27. Test Case 2: Thrust Profile for Costate Method

B-3 Earth-to-Jupiter Test Case

Figure 28. Earth-to-Jupiter Thrust Profile for Segmented Method
Figure 29. Earth-to-Jupiter Thrust Profile for Costate Method