2014

Automated trajectory design for impulsive and low thrust interplanetary mission analysis

Samuel Arthur Wagner
Iowa State University

Automated trajectory design for impulsive and low thrust interplanetary mission analysis

by

Samuel Arthur Wagner

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Aerospace Engineering

Program of Study Committee:

Bong Wie, Major Professor

John Basart

Ran Dai

Ping Lu

Ambar Mitra

Iowa State University

Ames, Iowa

2014

DEDICATION

I would like to thank my parents and all my family and friends who have helped me throughout my time as a student. Without their support I wouldn't have been able to make it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Bong Wie for his guidance and support throughout this research and the writing of this thesis. I would also like to thank my committee members for their efforts and contributions to this work: Dr. John Basart, Dr. Ran Dai, Dr. Ambar Mitra, and Dr. Ping Lu.

CHAPTER 1. INTRODUCTION

This dissertation describes a hybrid optimization algorithm that is able to determine optimal trajectories for many complex mission analysis and design orbital mechanics problems. This new algorithm will be used to determine optimal trajectories for a variety of mission design problems, including asteroid rendezvous, multiple gravity-assist (MGA), multiple gravity-assist with deep-space maneuvers (MGA-DSM), and low-thrust trajectory missions. The research described here was conducted at the Asteroid Deflection Research Center (ADRC) at Iowa State University.

1.1 Interplanetary Trajectories

Since the launch of Sputnik 1, on Oct. 4th, 1957, the human race has been driven by curiosity to explore beyond Earth and enter into the solar system. The first successful interplanetary probe, which passed within 22,000 miles of Venus on Dec. 14th, 1962, was NASA's Mariner 2 spacecraft. Since then a myriad of spacecraft have been launched to further explore the solar system and the interstellar space near our solar system. To continue exploring the solar system ever larger spacecraft, with increasingly complex scientific payloads, will be required. These missions require increasingly complex trajectories, due to the energy required for missions to the outer solar system and nearby interstellar space. The energy required to for such missions is often measure by the change in velocity, ΔV, required for the spacecraft to complete the mission.

To reach nearby destination such as the moon, Venus, Mars, and many near-Earth objects the required ΔV can often be directly obtained from the spacecraft or launch vehicle with traditional chemical propulsion systems. However, to reach further destinations, such as Mercury,

the outer planets, and asteroid and comets that are further away, a very large total ΔV is required. This required ΔV can currently be accomplished in three ways, including developing and launching heavy lift launch vehicles, utilizing planetary flybys, and with low-thrust solar electric propulsion systems.

The first method to obtain high ΔV's is to develop and launch larger launch vehicles, allowing high launch energies, known as C_3, or for the spacecraft to hold more propellant. This approach has the potential of allowing direct launches to destinations such as Jupiter and Saturn, but the cost of the launch vehicle development and manufacturing quickly becomes economically infeasible. Even with larger launch vehicles the limits of traditional chemical propulsion systems are quickly reached, because current chemical propulsion technology is nearly as efficient as possible, with maximum specific impulses, I_{sp}, in the 500-600 second range.

1.2 Gravity-Assist Trajectories

The second method to obtain the required ΔV is to utilize planetary flyby maneuvers, or gravity-assists. This allows spacecraft to leverage speed increases obtain from planetary flybys to reach further destinations. To understand the significance of gravity-assists and their importance to interplanetary missions, it is important to understand the classical theories of space travel, as developed by notable rocket pioneers such as Hohmann, Goddard, and Tsiolkovsky. Prior to the invention of gravity-assists, all spacecraft relied strictly on chemical propulsion as a means to obtain ΔV's. Because of the high ΔV's required for missions to Mercury and the outer solar system only missions to our nearby neighbors, Earth's Moon, Mars, Venus, and near-Earth asteroids, were possible.

In the 1950s a great deal of research to advance interplanetary trajectory design was performed by many notable orbital mechanics specialists such as Breakwell, Battin, Lawden, and numerous others [1–5]. In 1959 Battin even went so far as to compute round trip free return trajectories to Mars using solutions to Lambert's problem. Unfortunately he was unable to mathematically represent the 3-dimensional flyby trajectory because the mathematical representations to do this hadn't been developed yet. In 1961 Minovitch [6] developed a method to

represent the 3-dimensional geometry of these planetary flybys and was able to develop what is now known now as the patched conic method.

Since this development gravity-assist trajectories have been used by many missions, starting with the Mariner 10 mission. Other missions that have utilized gravity assist trajectories include missions such as Pioneer 10, Pioneer 11, Voyager 1, Voyager 2, Galileo, Cassini, and Messenger. With the Voyager and Pioneer missions gravity-assists trajectories were even used to obtain solar system escape trajectories. Details of the problem geometry and common gravity-assist trajectory models are discussed in Chapter 5.

1.3 Low-Thrust Interplanetary Trajectories

The last method currently utilized to obtain high ΔV for complex trajectories is with constant thrust trajectories. These trajectories utilize efficient solar electric propulsion technologies, which have much higher exhaust velocity than traditional chemical propulsion systems allow. However, with current technology solar electric propulsion systems are unable to achieve the high thrust force that traditional chemical propulsion affords. Because of the this thrust force limitation solar electric propulsion systems often have to be powered for days to months at a time, resulting in what is known as continuous low-thrust trajectories.

Recent advances in ultra-lightweight solar sails and high power solar electric propulsion systems have stimulated a renewed interest in continuous low-thrust trajectories. For interplanetary trajectories these efficient propulsion systems can be used to significantly increase the mass delivered to the destination. When compared to traditional chemical propulsion systems, solar electric thrusters have a much greater efficiency due to the increase in I_{sp}, typically around the 1000-3000 s range.

Prior to the 1990's interplanetary solar electric propulstion systems were not available. Since then these propulsion systems have been utilized on missions such as NASA's Deep Space 1 and Dawn missions and JAXA's Hayabusa mission. In this dissertation a method to model low-thrust trajectories is developed in Chapter 7. This model is then used to find optimal trajectories for an Asteroid Redirect Mission (ARM), which have the express purpose of returning a captured asteroid to the Earth.

1.4 Interplanetary Mission Analysis and Design

Determining trajectories for both impulsive multiple gravity-assist and low-thrust interplanetary mission presents significant optimization challenges. These problems are extremely nonlinear, have strong basins of attraction, and are discontinuous, when the planetary gravity-assist order is considered. Traditional methods for solving multiple gravity-assist trajectories often rely on the mission designer to prune the decision space, in an effort to obtain acceptable solutions. This becomes difficult, even for the most experience mission designer, when a large number of gravity-assists or deep-space maneuvers are require for the mission to be feasible. A review of commonly used mission design optimization algorithms are discussed in Chapter 4, as well as the framework of the hybrid genetic-nonlinear programming (GNLP) optimization algorithm. This algorithm is then used to determine optimal trajectories for both the impulsive and low-thrust missions presented in this dissertation.

1.4.1 Lambert's Problem

The problem of determining an orbit from two position vectors and a specified time of flight, know as Lambert's problem, is one of the most fundamental problems in astrodynamics [7–15]. These solutions were historically used to calculated the trajectories of comets, but today they are fundamental to the initial orbit determination problem. Because of their importance, efficient and robust solutions to Lambert's problem are necessary for nearly every interplanetary mission design problem. Solutions to Lambert's problem are used for both the MGA and MGA-DSM model. In addition, solutions to Lambert's problem are often used to determine the time-of-flight "initial guesses" for low-thrust trajectory optimization algorithms. A review of four of the most robust and efficient algorithm are discussed in Chapter 2. In order to determine the best Lambert solution algorithm to use with the MGA, MGA-DSM, and low-thrust models the computational performance for each solution is accessed on both central processing units and graphical processing units. Through this comparison an extremely robust Lambert's solution algorithm can be developed. This algorithm is crucial to minimizing the computational cost of all of the missions developed in this dissertation.

CHAPTER 2. COMPUTATIONAL SOLUTIONS TO LAMBERT'S PROBLEM

This chapter presents a comprehensive review of the four most common and robust solutions to Lambert's problem, which will be used for both the high and low thrust mission optimization problems. The four solutions to Lambert's problem discussed in this chapter are commonly known as Gooding's method, Battin's method, Sun's method, and the universal variable method. The performance of each Lambert solution method is evaluated on both central processing units (CPUs) and graphical processing units (GPUs), in an effort determine which solution algorithms are the most robust and efficient. Determining which solution(s) to Lambert's problem are both robust and efficient for various orbit types is important because all of the subsequent mission design and analysis in this dissertation utilizes hundreds of thousands to millions of solutions during the optimization process.

2.1 Introduction

The problem of determining an orbit from two position vectors and a specified time of flight, known as Lambert's problem, is one of the most fundamental problems in astrodynamics [7–15]. Historically, solutions were needed to calculate the orbital elements of planets, comets, and other solar system bodies from observations. In the 18th century, the problem of computing a comet's trajectory from observations was a topic of discussion by nearly every eminent astronomer and mathematician. John Henry Lambert (1728-1777) proposed that the orbit depends only on the chord, sum of the two radii, and semi-major axis, which is now known as Lambert's theorem. With the advent of human spaceflight (including robotic spaceflight) and the importance to

preliminary orbit determination, solutions to Lambert's problem have continued to be an area of active research interest in recent decades.

Today, solutions to Lambert's problem are used extensively in the orbital targeting problem, specifically in areas such as rendezvous analysis, missile targeting algorithms, and preliminary orbit determination. With the growing complexity of robotic space missions, such as NASA's Cassini, Galileo, Messenger, and OSIRIS REx missions, it is necessary to determine efficient and robust Lambert solution algorithms. In this chapter, four modern Lambert solution methods, with any necessary modifications for the CUDA GPU computing architecture requirements, will be examined for efficiency, robustness, and accuracy of the solutions.

With the advent of modern GPUs, massively parallel computations are now more widely accessible, enabling tens of millions of solutions to Lambert's problem to be evaluated per second. These computational capabilities will allow increasingly complex missions to be designed and flown in the future without the need for expensive supercomputers. Several solution implementations to Lambert's problem have been examined to determine the most efficient and robust algorithms for the purpose of preliminary mission analysis. For this study, all of the algorithms have been developed using Fortran and CUDA Fortran [16].

With the given relationship among the sum of the two radius vectors, chord, semi-major axis, and time-of-flight, Lambert's problem is well suited for initial orbit-determination searching techniques. In the algorithms presented, it is convenient to replace the sum of the two radius and the chord with the actual initial and final radius vectors. These are typically determined as a function of time. Various solutions to Lambert's problem can be found throughout literature [12–14, 17]. While numerous solution methods have been proposed, only four modern formulations will be tested and evaluated in this chapter. The methods evaluated in this chapter are the classical universal variable method [12, 14, 17], Battin's alternate approach to Gauss' original method [13, 14], as well as formulations by Gooding [9, 10] and Sun [18]. These methods were chosen because they represent the culmination of the last few decades of research and are among the most robust solution algorithms.

The search for preliminary mission trajectories often requires Lambert's problem to be solved tens of millions of times. The object of this chapter is to determine the most efficient

and robust solution implementations. This includes initial guess generations schemes and root-finding methods for methods for the Sun and universal variable methods. Because individual solutions to Lambert's problem are completely independent, the initial orbit determination problem lends itself well to parallel programming, especially when implemented directly on modern GPUs.

Before examining the specifics of any particular method, a basic introduction to the problem is presented. The initial and final radius vectors and time-of-flight are given respectively as, $\vec{r}_1$, $\vec{r}_2$, and Δt, with the two radius vector magnitudes represented as r_1 and r_2. The chord, c, and semiperimeter, s, are also needed for each of the solution methods. The chord is simply the distance between the two radius vectors, while the semiperimeter is half the sum of the triangle formed by the chord and radius vectors, defined as follows:

$$c = \mid \vec{r}_2 - \vec{r}_1 \mid = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos\theta} \tag{2.1}$$

$$s = \frac{r_1 + r_2 + c}{2} \tag{2.2}$$

A method to determine the transfer angle θ without quadrant ambiguity is described below [17]. This method replaces the standard long way and short way terminology that is often used when describing solutions to Lambert's problem. The transfer angle θ for a prograde orbit is determined as follows:

$$\theta = \begin{cases} \cos^{-1}\left(\dfrac{\vec{r}_1 \cdot \vec{r}_2}{r_1 r_2}\right) & \text{if } (\vec{r}_1 \times \vec{r}_2)_k \geq 0 \\ 360^o - \cos^{-1}\left(\dfrac{\vec{r}_1 \cdot \vec{r}_2}{r_1 r_2}\right) & \text{if } (\vec{r}_1 \times \vec{r}_2)_k < 0 \end{cases} \tag{2.3}$$

where the subscript k indicates the out-of-plane component of the cross product. Similarly, the transfer angle for a retrograde orbit is determined by the following equation:

$$\theta = \begin{cases} \cos^{-1}\left(\dfrac{\vec{r}_1 \cdot \vec{r}_2}{r_1 r_2}\right) & \text{if } (\vec{r}_2 \times \vec{r}_1)_k < 0 \\ 360^o - \cos^{-1}\left(\dfrac{\vec{r}_1 \cdot \vec{r}_2}{r_1 r_2}\right) & \text{if } (\vec{r}_1 \times \vec{r}_2)_k \geq 0 \end{cases} \tag{2.4}$$

The time to traverse an arc between two points is related to the transfer orbit by Kepler's time equation. From Kepler's equation, Lagrange developed a proof of Lambert's theorem. The

proof, which is briefly summarized here, has been the basis of nearly every proposed solution to Lambert's problem. Lagrange's equation removes the orbit eccentricity from Kepler's time-of-flight equation and is only a function of $r_1 + r_2$, c, and the semi-major axis, a, as given by

$$\sqrt{\frac{\mu}{a^3}}\Delta t = (\alpha - \beta) - (\sin\alpha - \sin\beta) \tag{2.5}$$

The two angular parameters, α and β, are defined in terms of the two physical parameters, c, s, and the semi-major axis, a, as follows:

$$\sin\frac{\alpha}{2} = \pm\sqrt{\frac{s}{2a}} \tag{2.6}$$

$$\sin\frac{\beta}{2} = \pm\sqrt{\frac{s-c}{2a}} \tag{2.7}$$

The quadrant ambiguities for Lagrange's alpha and beta parameters are resolved with the following restrictions:

$$\begin{aligned} &0 \le \alpha \le 2\pi, \quad 0 \le \beta \le \pi \quad \text{for} \quad \theta \le \pi \\ &0 \le \alpha \le 2\pi, \quad -\pi \le \beta \le 0 \quad \text{for} \quad \theta \ge \pi \end{aligned}$$

2.2 Solution Methods to Lambert's Problem

Solutions to Lambert's problem are formulated using Lagrange's time-of-flight equation, Eq. (2.5). In this section, the four Lambert solutions are introduced, including any changes necessary for the CUDA programming model. For the case of the universal variable and Sun method, this includes initial guess and root finding routines to improve algorithm efficiency and robustness over other solutions commonly found in the literature.

2.2.1 Battin's Method

The first Lambert solution method considered, commonly known as Battin's method, was first proposed in the 1980s and is perhaps the most mathematically rigorous algorithm [13]. This

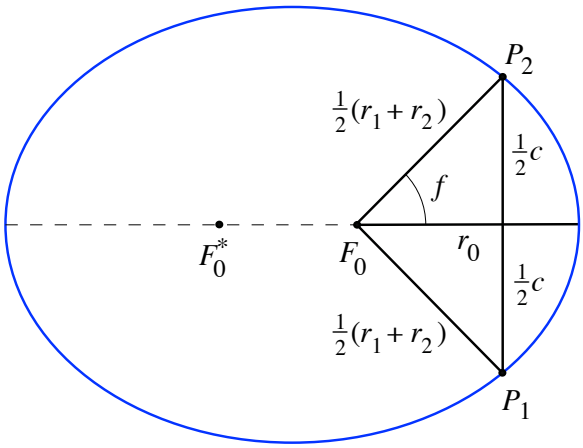


Figure 2.1 Transformed elliptical orbit necessary for the Battin's Lambert solution method.

solution is similar to Gauss' original solution except that it moves the singularity from 180 to 360 degrees and dramatically improves convergence when θ is large. Just as Gauss' solution does, Battin's method works for all types of orbits (elliptical, parabolic, and hyperbolic). Throughout this section, a brief introductions to Battin's method will be outlined. Any details left out can be found in [13].

Lambert's problem states that the time-of-flight is a function of a, r_1+r_2, and c. Therefore, the orbit can be transformed to any shape desired as long as the semi-major axis a, r_1+r_2, and c are held constant. The orbit is transformed such that the semi-major axis is perpendicular to the line from $\vec{r}_1$ to $\vec{r}_2$, which is c by definition [11, 13, 19]. The transformed ellipse is shown in Fig. 2.1. Further explanation of the transformed ellipse can be found in [13].

The first step in Battin's formulation is to define two non-dimensional parameters that are functions only of the problem geometry. The non-dimensional Lambert parameter, λ, to describe the problem geometry is

$$\lambda = \frac{\sqrt{r_1 r_2}}{s}\cos\frac{\theta}{2} = \pm\sqrt{\frac{s-c}{s}} \tag{2.8}$$

where, $-1 < \lambda < 1$. The sign ambiguity in the square root can be resolved as: $\lambda > 0$ for $0 < \theta < \pi$ and $\lambda < 0$ for $\pi < \theta < 2\pi$.

The dimensionless time-of-flight parameter for the algorithm is

$$T = \sqrt{\frac{8\mu}{s^3}}\Delta t \tag{2.9}$$

With this transformation, the equation for the pericenter radius equal to the main point, r_0 is described as

$$r_0 = a(1 - e_0) = r_{0p}\sec^2\frac{1}{4}(E_2 - E_1) \tag{2.10}$$

where e_0 is the eccentricity of the transformed orbit and r_{0p} is the mean point of the parabolic orbit from P_1 to P_2. The mean point of the parabolic radius r_{0p} is also given by

$$r_{0p} = \frac{1}{4}(r_1 + r_2 + 2\sqrt{r_1 r_2}\cos\frac{\theta}{2}) = \frac{1}{4}s(1+\lambda)^2 \tag{2.11}$$

Battin then introduces two new variables, ℓ and m, which are always positive and depend only on the problem geometry, as follows:

$$\ell = \left(\frac{1-\lambda}{1+\lambda}\right)^2 \tag{2.12}$$

$$m \equiv \frac{\mu\Delta t^2}{8r_{0p}^3} = \frac{T^2}{(1+\lambda)^6} \tag{2.13}$$

In its final form, Kepler's time-of-flight equation is transformed to

$$y^3 - (1 + h_1)y^2 - h_2 = 0 \tag{2.14}$$

The variable y is defined as a function of the problem parameters ℓ, x, and m, as follows:

$$y^2 \equiv \frac{m}{(\ell + x)(1 + x)} \tag{2.15}$$

The independent variable, x, is then determined from y for all orbit types as

$$x = \sqrt{\left(\frac{1-\ell}{2}\right)^2 + \frac{m}{y^2}} - \frac{1+\ell}{2} \tag{2.16}$$

The flattening parameters, h_1 and h_2, are functions of ℓ, m, x, and the continued fraction $\xi(x)$ defined as

$$h_1 = \frac{(\ell + x)^2(1 + 3x + \xi(x))}{(1 + 2x + \ell)[4x + \xi(x)(3 + x)]} \tag{2.17}$$

$$h_2 = \frac{m(x - \ell + \xi(x))}{(1 + 2x + \ell)[4x + \xi(x)(3 + x)]} \tag{2.18}$$

The function $\xi(x)$, needed for the calculation of h_1 and h_2, is defined by the following continued fraction:

$$\xi(x) = \cfrac{8(\sqrt{1+x}+1)}{3 + \cfrac{1}{5 + \eta + \cfrac{\frac{9}{7}\eta}{1 + \cfrac{c_\eta \eta}{1 + \cfrac{c_\eta \eta}{1 + \ldots}}}}} \tag{2.19}$$

where η is defined as

$$\eta = \frac{x}{(\sqrt{1+x}+1)^2}, \qquad -1 < \eta < 1 \tag{2.20}$$

and c_η is

$$c_\eta = \frac{n^2}{(2n)^2 - 1}, \qquad n = 4, 5, \ldots \tag{2.21}$$

2.2.1.1 Solving the cubic function

The largest real root of Eq. (2.14) must be found. A successive substitution algorithm can be used to solve for the largest root of y and determine the new x through the definition in Eq. (2.16). The first step is to calculate B and u as follows:

$$B = \frac{27 h_2}{4(1 + h_1)^3} \tag{2.22}$$

$$u = \frac{B}{2(\sqrt{1 + B} + 1)} \tag{2.23}$$

A second continued fraction expansion, $K(u)$, is also defined by Battin to determine the final solution of y, as follows:

$$K(u) = \cfrac{\frac{1}{3}}{1 + \cfrac{\frac{4}{27}u}{1 + \cfrac{\frac{8}{27}u}{1 + \cfrac{\frac{2}{9}u}{1 + \cfrac{\frac{22}{81}u}{1 + \ddots}}}}} \tag{2.24}$$

The odd and even coefficients are obtained from the following two equations:

$$\gamma_{2n+1} = \frac{2(3n+2)(6n+1)}{9(4n+1)(4n+3)} \tag{2.25}$$

$$\gamma_{2n} = \frac{2(3n+1)(6n-1)}{9(4n-1)(4n+1)} \tag{2.26}$$

The largest positive real root for the cubic equation is then found as

$$y = \frac{1+h_1}{3}\left(2 + \frac{\sqrt{1+B}}{1+2u(K^2(u))}\right) \tag{2.27}$$

The solution to the cubic function, Eq. (2.14), is solved through a successive substitution method. When an x value is converged upon, the semi-major axis of the orbit can be found. With the semi-major axis, $a = \mu(\Delta t)^2/16r_{op}^2xy^2$, the initial and final velocities can be easily calculated using the standard Lagrange coefficients f, g, $\dot{f}$, and $\dot{g}$.

The initial conditions for x that give guaranteed convergence are:

$$x_0 = \begin{cases} 0 & \text{parabola, hyperbola} \\ \ell & \text{ellipse} \end{cases} \tag{2.28}$$

The dimensionless parabolic time-of-flight needed to assign the initial condition is a function of the Lambert parameter defined as

$$T_p = \frac{4}{3}(1-\lambda^3) \tag{2.29}$$

Battin's method to determine the orbit's semi-major axis from the variables λ and T is stated as below. It should be noted that CUDA Fortran [16] does not allow recursive algorithms. Therefore, all continued fractions are always calculated out to 20 total fraction levels. This level of continued fractions provide sufficient numerical accuracy when compared with other Lambert solution algorithms, typically within a small margin of error (less than 10^{-8}).

1. Compute the dimensionless parameters ℓ, m, and r_{0p}.
2. From the parabolic time-of-flight, T_p, determine x_0.
3. In the following order calculate: η, ξ, h_1, h_2, B, u, $K(u)$.
4. Compute the solution for y from Eq. (2.27) and the updated x from Eq. (2.16).
5. Go to step 3 and repeat until the desired tolerance for the change in x is met or the maximum number of iterations is exceeded.
6. Output the converged semi-major axis, a, or the initial and final velocities with the Lagrange coefficients.

2.2.2 Gooding's Method

The second Lambert solution method considered is known as Gooding's method [10, 20]. This method is an extension of a Lancaster and Blanchard's unified form of Lambert's theorem developed in the 1960s[9]. Gooding was able to formulate robust initial guesses as well as derivatives for a high order root finding routine. The final version of the algorithm developed for this study differs from the algorithms presented in Gooding's papers [10, 20], but can provide approximately 100% convergence. This method will also be shown to be one of the most computationally efficient algorithms.

For this solution method, two necessary parameters, T and q, are defined in the same way as, T and λ, in Battin's method. However, the choice of the independent universal variable differs. In this case the independent universal variable refers to a choice in variables in which

the resulting solution method applies to both elliptical and hyperbolic orbits. The sign of q is chosen the same as λ, as

$$q = \lambda = \frac{\sqrt{r_1 r_2}}{s} \cos\frac{\theta}{2} \tag{2.30}$$

$$T \equiv \sqrt{\frac{8\mu}{s^3}} \Delta t \tag{2.31}$$

The independent variable for this method is defined as

$$x^2 = 1 - \frac{s}{2a} \tag{2.32}$$

Elliptical orbits cover the range from -1 to 1, with a value of 0 corresponding to the minimum energy solution. An x value of exactly 1 represents a parabolic orbit, while x values larger than 1 are hyperbolic orbits.

Two additional variables, K and E, that are functions of the problem geometry are defined as

$$K = q^2 = 1 - \frac{c}{s} \tag{2.33}$$

$$E(x) = x^2 - 1 = -\frac{s}{2a} \tag{2.34}$$

2.2.2.1 Formulation of the elliptical and hyperbolic time equation

The following time-of-flight equation formulation works for both elliptical and hyperbolic orbits. However, this formulation does not work well when the orbit is near parabolic. For near parabolic orbits a separate formulation is necessary. The following equations allow the time equation to be represented in terms of the independent variable, x, and the physical parameters of the problem.

$$y = \sqrt{|E|} \tag{2.35}$$

$$z = \sqrt{1 + KE} \tag{2.36}$$

$$f = y\,(z - qx) \tag{2.37}$$

$$g = xz - qE \tag{2.38}$$

$$d = \begin{cases} \text{atan}(f/g), & E < 0 \\ \text{atanh}(f/g), & E > 0 \end{cases} \tag{2.39}$$

The inverse tangent function should be computed unambiguously, typically by the ATAN2 function. The time-of-flight function for elliptical and hyperbolic orbits for the root-finding routine is then defined as follows:

$$F(x) = \frac{2}{E}\left(x - qz - \frac{d}{y}\right) - T = 0 \tag{2.40}$$

To solve for x for a corresponding required time parameter, T, a second order Halley iterative method is used [21]. Halley's method requires second order derivative of the time equation with respect to the independent variable x. This high-order method requires more computations, when compared with the commonly used Newton-Raphson methods, but significantly increases the rate of convergence. In addition, initial guess conditions are much more relaxed when compared to low order root finding methods. Halley's method for solving $F(x) = 0$ for x is given by

$$x_{n+1} = x_n - \frac{2F_nF'_n}{2\left(F'_n\right)^2 - F_nF''_n} \tag{2.41}$$

The first and second derivatives of $F(x)$ with respect to x are given by

$$F' = -\frac{3x(F+T) + 4q^3\dfrac{x}{z} - 4}{E} \tag{2.42}$$

$$F'' = -\frac{3(F+T) + 5xF' + 4\left(\frac{q}{z}\right)^3\left(1+q^2\right)}{E} \tag{2.43}$$

where T is the specified time-of-flight.

2.2.2.2 Formulation of the time equation for near parabolic orbits

For near parabolic orbits, in the neighborhood of $x \approx 1$, Eq. (2.40) suffers from a loss of significant digits. For near parabolic orbits the Gooding time-of-flight equation can be written as

$$F(x) = \sigma(-E) - qK\sigma(-KE) - T = 0 \tag{2.44}$$

where $\sigma(u)$ has the form:

$$\sigma(u) = 2\frac{\sin^{-1}\sqrt{u} - \sqrt{u}\sqrt{1-u}}{\sqrt{u^3}} \tag{2.45}$$

Equation (2.45) can be represented by a series function, as done by Gooding's original algorithms. However, series-based solutions are more difficult to implement, and there is no significant increase in the algorithm's performance when the transcendental form is directly implemented, likely due to compiler optimization. For this reason, all the necessary derivatives for the algorithm are kept in the transcendental form. The first and second order derivatives for Eq. (2.45) are then given by

$$\sigma' = -\frac{u}{\sqrt{1-u}} + \frac{3\sin^{-1}\sqrt{u}}{\sqrt{(u)}} - \frac{3}{u^2\sqrt{1-u}} \tag{2.46}$$

$$\sigma'' = \frac{15\sin^{-1}\sqrt{u}\sqrt{(1-u)^3} - 15\sqrt{u} + 20\sqrt{u^3} - 3\sqrt{u^5}}{2\sqrt{u^7}\sqrt{(1-u)^3}} \tag{2.47}$$

It is still necessary to calculate the derivatives of the near parabolic time equation, which are given as

$$F' = -2x\left(\sigma'(-E) - q^5\sigma'(-KE)\right) \tag{2.48}$$

$$F'' = \frac{F'}{x} + 4x^2\left(\sigma''(-E) - q^7\sigma''(-KE)\right) \tag{2.49}$$

2.2.2.3 Initial guess generation

With the time equation formulation and all necessary derivatives completed for all orbit forms, the last step is to determine initial guesses. By examining Fig. 2.2, it is clear that the time curves are strictly monotonically decreasing, except for the case when q is exactly 1. An initial guess generation scheme, based on a bi-linear approximation, was developed by Gooding. This initial guess generation method is extremely robust and is outlined below.

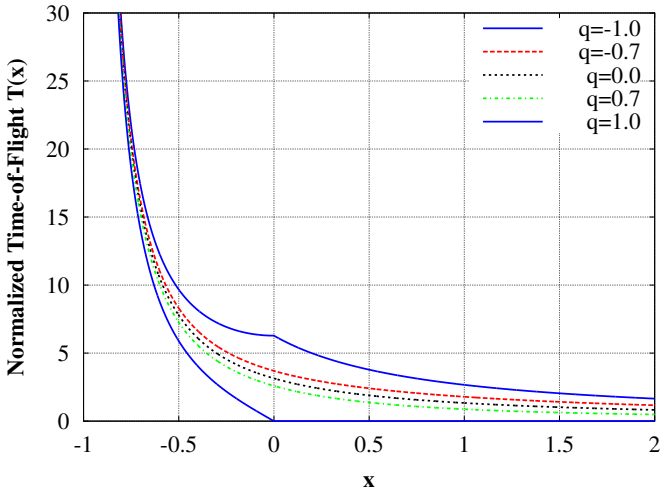


Figure 2.2 Plot of $T(x)$ vs x for Gooding's method.

The first step in determining the initial guess is to determine the sign of the solution, x. The sign is determined by evaluating the time-of-flight from Eq. (2.40) at $x = 0$. If the desired time, T, is less than T_0, the initial guess is obtained as

$$x_0 = T_0\frac{T_0 - T}{4T} \tag{2.50}$$

However, in the case that the desired time is greater than T_0, the following equations are necessary for the initial guess:

$$x_{01} = -\frac{T - T_0}{T - T_0 + 4} \tag{2.51}$$

$$W = x_{01} + c_1\sqrt{2 - \frac{\theta}{\pi}} \tag{2.52}$$

$$x_{03} = \begin{cases} x_{01} & , \quad W \geq 0 \\ x_{01} + (-W)^{\frac{1}{16}}\left(x_{02} - x_{01}\right) & , \quad W < 0 \end{cases} \tag{2.53}$$

$$\lambda = 1 + c_1 x_{03}\frac{4}{4 - T - T_0} - c_2 x_{03}^2\sqrt{1 + x_{01}} \tag{2.54}$$

$$x_0 = \lambda x_{03} \tag{2.55}$$

This final equation for the initial guess it designed to give robust initial guesses even when the solution is close to -1. The following constants were determined through the numerical study: $c_1 = 0.5$, $c_2 = 0.03$.

The basics of the final algorithm are provided below. The inputs to the algorithm are q and T. The algorithm developed in this section converges for approximately 100% of solutions and is also one of the most efficient of the four algorithms. Gooding's method is outlined as

1. Evaluate T_0 when $x = 0$ and determine the initial guess from Eqs. (2.50) or (2.55).
2. If x is close to 1.0, calculate E and K to evaluate F, F', and F'' from Eqs. (2.44), (2.48), and (2.49), respectively.
3. Otherwise calculate z, d, y, and E and evaluate F, F', and F'' from Eqs. (2.40), (2.42), and (2.43), respectively.
4. Update x using Halley's method.
5. Go to step 2 and repeat until the desired tolerance for the change in x is met or a maximum number of iterations is exceeded.
6. Output the converged semi-major axis, a, or the initial and final radius vectors.

2.2.3 Sun's Method with Improved Convergence

The Sun method [18] is among the most robust and efficient Lambert solvers available. The formulation utilized by Sun is similar to the Gooding algorithm. The choice of Lambert parameter, time parameter, and independent variable are comparable to formulation proposed by Lancaster and Blanchard [9]. However, the final form of the Lagrange time equation differs significantly. This method has been improved upon here, by utilizing the high-order Halley root-finding method and by an initial guess generation scheme that is simple and robust.

The time parameter, τ, for Sun's method is defined as

$$\tau = 4\Delta t\sqrt{\frac{\mu}{m^3}} = \Delta t\sqrt{\frac{2\mu}{s^2}} \tag{2.56}$$

It is worth noting that this scaled time is exactly half of the scaled time parameter of the Battin and Gooding methods.

The variable, m, is the sum of the two radii and the chord, which is also double the semiparameter, s, and is defined as follows:

$$m = r_1 + r_2 + c \tag{2.57}$$

The non-dimensional Lambert parameter, σ, is equivalent to the other Lambert parameters, λ and q. This parameter is describe by the problem geometry as

$$\sigma^2 = \frac{4r_1r_2}{m^2}\cos\frac{\theta}{2} \tag{2.58}$$

where, $-1 < \sigma < 1$. The sign ambiguity in the square root can be resolved as: $\sigma > 0$ for $0 < \theta < \pi$ and $\sigma < 0$ for $\pi < \theta < 2\pi$.

The independent variable for Sun's method is defined as

$$x^2 \equiv 1 - \frac{m}{4a} = 1 - \frac{s}{2a} \tag{2.59}$$

The parabolic and minimum energy solution, necessary for initial guesses, occur when x has a value of 1 and 0, respectively, as

$$\tau_p = \frac{2}{3}\sqrt{1-\sigma^3} \tag{2.60}$$

$$\tau_{ME} = \cos^{-1}\sigma + \sigma\sqrt{1-\sigma^2} \tag{2.61}$$

2.2.3.1 Time-of-flight and derivative functions

Using Sun's notation, the Lagrange time-of-flight equation can be expressed in terms of the sum of two transcendental equations, as

$$F(x) = \phi(x) - \phi(y) - \tau \tag{2.62}$$

where τ is the specified time-of-flight and y is defined in terms of x as

$$y = \begin{cases} \sqrt{1-\sigma^2(1-x^2)} & \text{if} \quad \sigma > 0 \\ 1 & \text{if} \quad \sigma \approx 0 \\ -\sqrt{1-\sigma^2(1-x^2)} & \text{if} \quad \sigma < 0 \end{cases} \tag{2.63}$$

The transcendental function, ϕ, has the form

$$\phi(u) = \cot^{-1}\frac{u}{\sqrt{1-u^2}} - \frac{1}{3u}\left(2+u^2\right)\sqrt{1-u^2} \tag{2.64}$$

The time of flight equation for all possible orbit cases then becomes

$$F(x) = \begin{cases} \dfrac{1}{(1-x^2)^{\frac{3}{2}}}\left(\cot^{-1}\dfrac{x}{\sqrt{1-x^2}} - \cot^{-1}\dfrac{y}{\sqrt{1-y^2}}\ldots\right. & \\ \left.-x\sqrt{1-x^2} + y\sqrt{1-y^2}\right) - \tau & \text{if} \quad |\,x\,| < 1 \\ \dfrac{1}{(1-x^2)^{\frac{3}{2}}}\left(-\coth^{-1}\dfrac{x}{\sqrt{x^2-1}} + \coth^{-1}\dfrac{y}{\sqrt{y^2-1}}\ldots\right. & \\ \left.+x\sqrt{x^2-1} - y\sqrt{y^2-1}\right) - \tau & \text{if} \quad x > 1 \\ \dfrac{2}{3}\left(1-\sigma^3\right) - \tau & \text{if} \quad x \approx 1 \end{cases} \tag{2.65}$$

where τ is the specified time-of-flight.

The two inversion circular functions for elliptical orbits are restricted to the following ranges:

$$0 \le \cot^{-1}\frac{x}{\sqrt{1-x^2}} \le \pi \quad , \quad \frac{-\pi}{2} \le \cot^{-1}\frac{y}{\sqrt{1-y^2}} \le \frac{\pi}{2} \tag{2.66}$$

With the range restrictions for the two circular functions, they can be redefined as

$$\cot^{-1}\frac{x}{\sqrt{1-x^2}} = \cos^{-1} x \tag{2.67}$$

$$\cot^{-1}\frac{y}{\sqrt{1-y^2}} = \tan^{-1}\frac{\sqrt{1-y^2}}{y} \tag{2.68}$$

The Halley root-finding method is used to iteratively solve $F(x) = 0$. As before, both first and second order derivatives of the Sun time-of-flight functions are required as follows:

$$F' = \frac{1}{1-x^2}\left[3x(F+\tau) - 2\left(1-\sigma^3\frac{x}{|\,y\,|}\right)\right] \tag{2.69}$$

$$F'' = \frac{1}{x(1-x^2)}\left[\left(1+4x^2\right)F' + 2\left(1-\sigma^5\frac{x^3}{|\,y\,|^3}\right)\right] \tag{2.70}$$

As before, the update scheme for Halley's method is described as follows:

$$x_{n+1} = x_n - \frac{2F_nF_n'}{2\left(F_n'\right)^2 - F_nF_n''} \tag{2.71}$$

An alternative Laguerre high-order root-finding method was also tested, which is expressed as

$$x_{n+1} = x_n - \frac{nF_n}{F_n' \pm \sqrt{|(n-1)^2F_n'^2 - n(n-1)F_nF_n''|}} \tag{2.72}$$

where, n, is a user defined positive integer value. If n is 1 this method breaks down to the classical Newton method. For Sun's method a value of 4 works well. The sign ambiguity is resolved by taking the negative of the sign of the derivative, F_n'. This method has been utilized by others for a high-order root-finding method to use in conjunction with solutions to Lambert's problem [22, 23]. However, when the Laguerre method is compared to Halley's method, the average number of iterations required by Sun's method is higher, resulting in an approximately 5.5% slower solution algorithm.

A simple scheme for initial guesses is used for this algorithm. By utilizing knowledge of the orbit, accurate initial guesses can be obtained. The initial guesses, combined with the high-

order Halley convergence method, provides an efficient and robust algorithm and are selected as

$$x_0 = \begin{cases} 0.5 & \text{if} \quad \tau < \tau_{ME} \quad \text{elliptical} \\ 0 & \text{if} \quad \tau \approx \tau_{ME} \quad \text{elliptical} \\ -0.5 & \text{if} \quad \tau > \tau_{ME} \quad \text{elliptical} \\ 1 & \text{if} \quad \tau \approx \tau_P \quad \text{parabolic} \\ 3.0 & \text{if} \quad \tau > \tau_P \quad \text{hyperbolic} \end{cases} \tag{2.73}$$

Sun's method is outlined below.

1. From σ calculate τ_p and τ_{ME} and determine the initial guess as described by Eq. (2.73).
2. Evaluate F, F', F'' from Eqs. (2.65), (2.69), and (2.70) respectively.
3. Update x using Halley's method.
4. Go to step 2 and repeat until the desired tolerance for the change in x is met or a maximum number of iterations is exceeded.
5. Output the converged semi-major axis, a, or the initial and final velocity vectors.

For this algorithm, the parabolic time equation is used only when x is within a tolerance of $\pm 10^{-8}$ of 1.0. The advantage that this algorithm has over the Battin and Gooding algorithm is its simplicity and ease of implementation. Combined with the given initial guesses, this algorithm is extremely robust and is a strong candidate for those working on mission optimization packages.

2.2.4 The Universal Variable Method

The last method tested in this study is commonly known as the universal variable method. This method has been one of the most extensively studied and published in recent decades [12, 14, 17, 23, 24]. The independent variable chosen for this solution works universally for all orbits. Unlike the Gooding and Sun methods, the variable selection and accompanying time-of-flight formulation applies to all orbits.

The universal variable, z, is defined as

$$z = \frac{x^2}{a} \tag{2.74}$$

Three new variables, A, y, and x are introduced as follows:

$$A = \sin\theta \sqrt{\frac{r_1 r_2}{1 - \cos\theta}} \tag{2.75}$$

$$y = r_1 + r_2 - A\frac{1 - zS}{\sqrt{C}} \tag{2.76}$$

$$x = \sqrt{\frac{y}{C}} \tag{2.77}$$

The Stumpff functions, $C(z)$ and $S(z)$, are required for this formulation. They were developed by the German astronomer Karl Stumpff and are commonly represented by a series function. However, it is convenient to represent them as trigonometric functions. If z is positive, the sin and cos functions are utilized; however, for hyperbolic orbits ($z < 0$) the corresponding hyperbolic functions must be used. For parabolic orbits, C and S have values of $\frac{1}{2}$ and $\frac{1}{6}$, respectively. The Stumpff functions are described by

$$C(z) = \begin{cases} \dfrac{1 - \cos\sqrt{z}}{z} & \text{if} \quad z > 0 \\ \dfrac{\cosh\sqrt{-z} - 1}{-z} & \text{if} \quad z < 0 \\ \dfrac{1}{2} & \text{if} \quad z = 0 \end{cases} \tag{2.78}$$

$$S(z) = \begin{cases} \dfrac{\sqrt{z} - \sin\sqrt{z}}{\sqrt{z^3}} & \text{if} \quad z > 0 \\ \dfrac{\sinh\sqrt{-z} - \sqrt{-z}}{\sqrt{-z^3}} & \text{if} \quad z < 0 \\ \dfrac{1}{6} & \text{if} \quad z = 0 \end{cases} \tag{2.79}$$

The time-of-flight function is defined as

$$F(z) = x^3 S + A\sqrt{y} - \sqrt{\mu}\Delta t \tag{2.80}$$

The first and second derivatives of the time-of-flight function with respect to z are also required as

$$F'(z) = x^3 S' + 3Sx^2 x' + \frac{Ay'}{2\sqrt{y}} \tag{2.81}$$

$$F''(z) = 6Sxx'^2 + 6x^2 x' S' + x^3 S'' + 3Sx^2 x'' + \frac{Ay''}{2\sqrt{y}} - \frac{Ay'^2}{4\sqrt{y^3}} \tag{2.82}$$

By examining Eq. (2.76), it can be seen that the universal variable formulation suffers from one major drawback. When θ is less than π, A is positive. If z becomes a large negative number, corresponding to orbits that become *too* hyperbolic, S and C become large positive numbers. In some cases, $\frac{A(1-zS)}{\sqrt{C}}$ will be greater than $r_1 + r_2$, which means Eq. (2.76) will become negative. This negative value results in an imaginary number in Eq. (2.77). This will cause the time equation to intersect with the $t = 0$ axis with orbits that are *too* hyperbolic. Any algorithm implementing this method should check for negative y values. When y becomes negative, the algorithm should be terminated.

An initial guess generation method is provided below. The first step in determining initial guesses is to determine the parabolic time-of-flight. For parabolic orbits, the universal variable, z, has a value equal to zero, and we have

$$T_p = \left. \frac{x(z)^3 S(z) + A\sqrt{y(z)}}{\sqrt{\mu}} \right|_{z=0} \tag{2.83}$$

The specified time-of-flight, Δt, can be compared with the parabolic time-of-flight to determine whether the orbit is elliptical, near parabolic, or hyperbolic. As shown in Fig. 2.3, the time curve is strictly monotonically increasing, so derivative-based root finding methods will be well behaved. For zero-revolution elliptical orbits, z has an upper bound of $4\pi^2$. An initial guess of $2\pi^2$ for elliptical orbits works well. If the orbit is near parabolic, an initial guess of 0 is used, while an initial guess of $-\frac{\pi}{32}$ works well for hyperbolic orbits. With the algorithm, shown below, convergence typically occurs in three to five iterations.

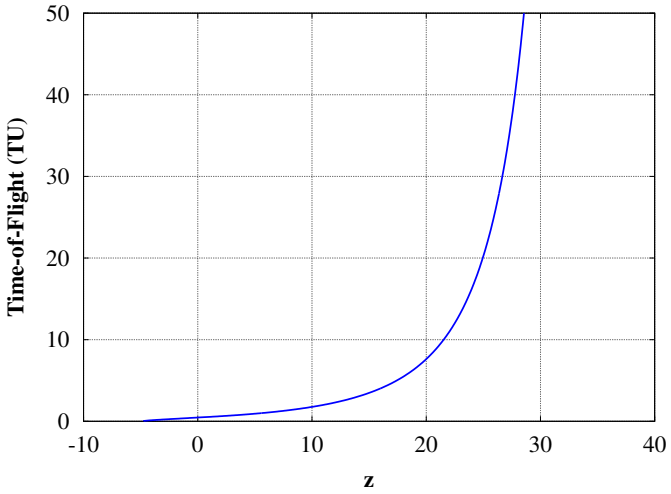


Figure 2.3 Time-of-flight (in scaled canonical units) versus z.

The universal variable method is outline as follows:

1. From the inputs, r_1, r_2, θ, and Δt, calculate A.
2. From the parabolic time-of-flight ($z = 0$), determine the initial guess.
3. Evaluate S, S', S'', C, C', C'', x, x', x'', y, y', y'' to calculate F, F', and F''.
4. Update z with Halley's method.
5. Go to step 3 and repeat until the desired tolerance for the change in z is met or a maximum number of iterations is exceeded.
6. Output the converged semi-major axis or the initial and final velocity vectors.

2.3 Results

For the four Lambert algorithms, two tests were developed to demonstrate the robustness and efficiency of each algorithm. The first test is designed to demonstrate the robustness of each algorithm. This is done by determining the number of iterations each algorithm requires to converge for a series of four test scenarios. The second test is designed to determine the efficiency of each algorithm, on both CPU and GPU computational architectures. This is done

by evaluating an example 25-year search for a direct mission to Mars. This mission scenario is representative of the type of initial orbit determination problem typically performed with Lambert searches. Details of the CPU and GPU workstations used to perform the tests can be found in Tables 2.1(a) and 2.1(b). The CPU tests were all run on a standard Dell workstation, while the GPU tests were performed on an Amax workstation with an NVIDIA Tesla C2050 GPU.

Table 2.1 Information on the workstations used to determine the computational efficiency of each Lambert algorithm.

(a) Workstation information.

	Workstation
Model	Dell T3500 Workstation
Operating System	Windows Vista Professional 64-bit
Processor	Intel(R) Xenon(R) W3520 2.67 GHz -4 core
Memory	6 GB 1066 MHz DDR3
GPU	NA

(b) GPU workstation information.

	GPU Workstation
Model	Amax ServMax Tesla GPU HPC
Operating System	RHEL v5.4
Processor	2x Intel Xeon LGA1366 2.66 GHz -6 core
Memory	32 GB 1333 MHz DDR3
GPU	4x Tesla C2050

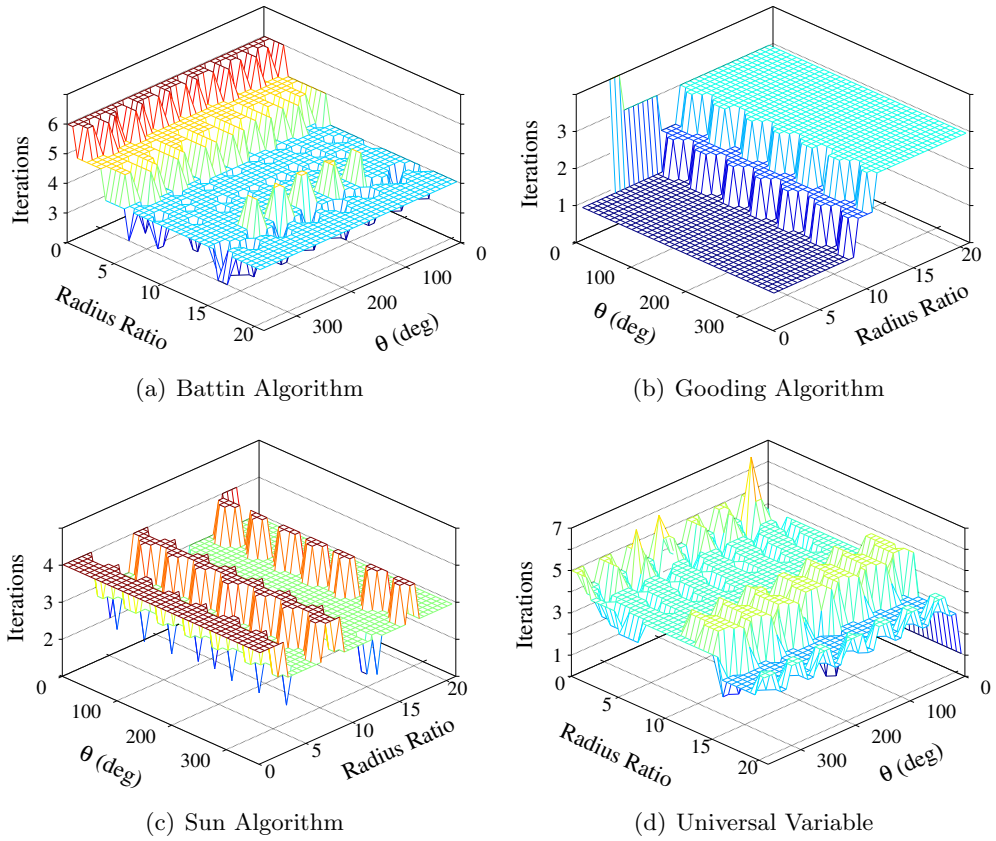
2.3.1 Demonstrating Robustness of Each Algorithm

It is useful to develop a test in which all four algorithms can be directly compared. The universal variable algorithm is not a function of the Lambert parameter (λ, q, or σ), which means that parameters directly from Lambert's theorem must be used for the tests.

To compare the robustness of each of the four algorithms, a series of four test cases were developed to simulate the types of missions solutions to Lambert's problem are often used for. The tests are designed to include all orbit types, from very hyperbolic to highly elliptical orbits. In each test case, an orbit around the Sun is assumed with r_2 held constant at 1 AU ($1\ AU = 149,599,650\ km$). The initial radius, r_1, is then varied from 0.1 to 20 AU while the

transfer angle, θ, is simultaneously varied from 0 to 360 degrees. Each of the four test cases is then distinguished by the required transfer times of 0.1, 2, 5, and 10 years, respectively. For Sun-centered orbits the sun gravitational parameter, μ, has a value of 132,712,440,018 km^2/s^3.

Each algorithm can then be compared by examining the number of iterations versus radius ratio, r_1/r_2, and transfer angle, θ. A comparison of the number of iterations required for each algorithm for the third test case, with a Δt of 5 years, is shown in Fig. 2.4. From these figures, it is apparent that the number of iterations required for each of the four algorithms varies very little with transfer angle. The same general trend is true for the other three test cases.



(a) Battin Algorithm

(b) Gooding Algorithm

(c) Sun Algorithm

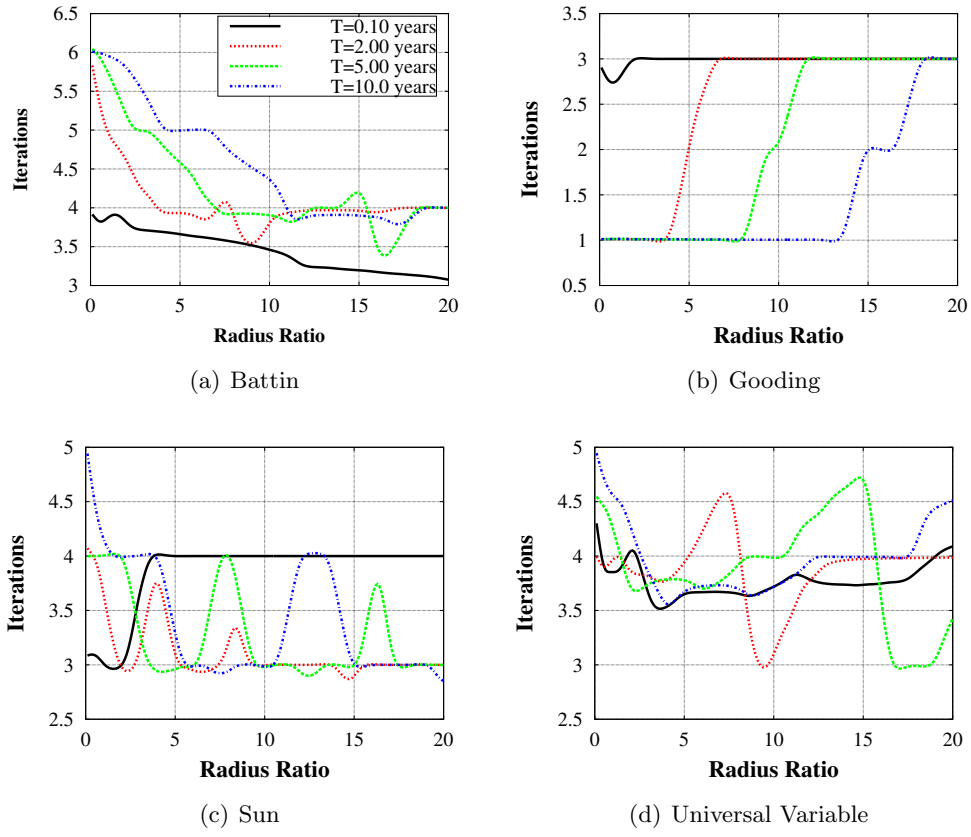
(d) Universal Variable

Figure 2.4 Comparison of various Lambert algorithms for the case 3 results.

With this knowledge it is convenient to compare the number of iterations required versus the radius ratio. This is accomplished by averaging the number of iterations for each radius ratio over the entire range of transfer angles. The results for all four test cases are shown in

Fig. 2.5. Several conclusions can be drawn for each solution method to Lambert's problem by examining this figure.

The Battin method is most efficient for short transfer times. As the plots show, the required number of iterations typically increases as the time-of-flight is increased. For each test case, the number of iterations required also decreases as the radius ratio increases. For the worst case scenario, large transfer times and small radius ratios, the algorithm takes up to six iterations to converge. For all other orbit combinations convergence typically occurs within three to five iterations. The average number of iterations required for each test case is shown in Table 2.2.



(a) Battin

(b) Gooding

(c) Sun

(d) Universal Variable

Figure 2.5 Comparison of various Lambert algorithms for the study case 3.

When comparing the number of required iterations, the Gooding algorithm has the most predictable results. Except for the case when the transfer angle was exactly 0 or 360 degrees, convergence always occurs within three iterations. This high convergence rate is due to the extremely accurate initial guess using the bilinear initial guess approximations. Unlike Battin's

Table 2.2 Average iterations required for the four test cases for each of the Lambert algorithms.

	Battin	**Gooding**	**Sun**	**Universal Variable**
Case 1	3.44	2.99	3.86	3.77
Case 2	4.06	2.51	3.11	3.88
Case 3	4.26	2.05	3.26	3.87
Case 4	4.54	1.45	3.39	3.98

method, this algorithm requires fewer iterations as the time of flight is increased. However, as the radius ratio increases, the number of iterations for the Gooding algorithm also increases. Out of the four Lambert algorithms, Gooding's algorithm is the most well behaved, resulting in extremely predictable performance.

Like Gooding's method, Sun's algorithm is well behaved. When solutions have large transfers times and small radius ratios, the algorithm typically converges within no more than four iterations. Except in cases where the initial guess is extremely close to the solution, this algorithm takes at least three iterations to converge. In the worst case scenario, this method takes no more than six iterations to converge on a solution.

The universal variable algorithm has the least predictable results when compared with the other algorithms. However, with the Halley root finding method, convergence typically occurs in at least five iterations. The exception is when y becomes negative or when θ is exactly 180 degrees. For all the test cases, the average number of iterations is approximately four, making this algorithm slightly more efficient than Battin's algorithm, but worse than the Gooding and Sun methods. Out of the four algorithms tested, the universal variable method has the largest range of required iterations

In practice, determining the number of iterations required for each algorithm is a good way to test each algorithm's robustness. However, a large problem in which Lambert's problem is solved tens to hundreds of millions of times is required to determine the overall efficiency of each algorithm. The second set of tests is designed to determine the computational efficiency of each algorithm on both CPU and GPU architectures.

2.3.2 Testing the Computational Efficiency of Each Algorithm

Solutions to Lambert's problem are typically used for initial orbit determination problems, which often require millions solutions at a time. Solutions can be evaluated in parallel because each solution to Lambert's problem is independent. In this case, both standard serial and parallel searches for an Earth to Mars transfer orbit are tested. All three versions of the algorithm execute a grid search of launch dates and times of flight, requiring approximately 250 million solutions. This search ensures saturation of the GPU while keeping CPU run times low enough for testing purposes.

Each of the three search implementations are written in Fortran or CUDA Fortran [16] and have nearly identical implementations. Keeping the search configurations as close as possible ensures that the efficiency of both the CPU and GPU algorithms can be directly compared. The first implementation is standard single-thread Fortran, the second is parallel Fortran with OpenMP [25], and the third is implemented using PGI's CUDA Fortran [16]. The GPU implementation was written to use a single GPU, in this case an NVIDIA Tesla C2050.

The final grid search was run over a 25-year period, with launch dates from 2025 to 2050 and a maximum time-of-flight of 1000 days. By using a grid search time step of 0.191 days exactly, 250,002,660 Lambert's problem solutions are required. The resulting pork chop plot of departure V_∞, with contour levels ranging from 3 to 9 km/s, is shown in Fig. 2.6.

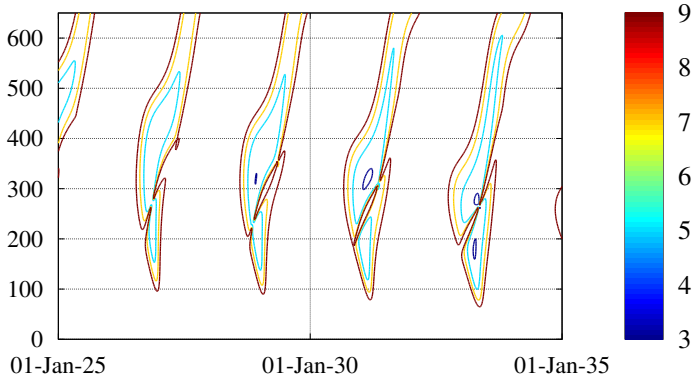


Figure 2.6 50-year porkchop plot of departure V_∞ for Mars mission.

The robustness of each algorithm can be determined by monitoring the number of times each algorithm fails to converge during the search. The initial orbit determination and orbit op-

Table 2.3 Number of failures for each Lambert algorithms for the Earth to Mars sample mission.

	# Failed Solutions	Failed %
Battin	0	0.00E-05
Gooding	0	0.00E+00
Sun	23	9.20E-06
Universal Variable	80,219	3.20E-01

timization problems require robust solutions to Lambert's problem. Therefore, only algorithms that are both extremely robust and efficient are suitable for such problems. From Table 2.3, it can be seen that the Battin, Gooding, and Sun algorithms nearly always converge on a solution, while the universal variable algorithm fails to converge approximately 0.32% of the time. A pure universal variable approach fails more often than this because y from Eq. (2.76) becomes negative for orbits that are *too* hyperbolic. However, this in a known limitation to the solution method. When this situation is encountered the universal variable algorithm uses the Gooding Lambert algorithm to determine a solution. For this test case negative y's were encountered approximately 2.4 million times, or approximately 0.9% of solutions. While this is a relatively small percentage, it is five orders of magnitude higher than the failure percentage of the Sun algorithm, which had the next highest failure rate. The Sun algorithm failed a total of 23 times, all of which were solutions with nearly exactly parabolic orbits, corresponding to solutions that have an x value very close to 1.0. This failure suggests that, as with Gooding's method, near parabolic orbits would improve with an alternative formulation for the time equation and the accompanying derivatives. Out of the four algorithms, only Battin's and Gooding's algorithms converged for 100% of solutions.

2.3.2.1 GPU performance

The CPU and GPU run times for each version of the search algorithm for all four Lambert algorithms are shown in Table 2.4. For the serial implementation, Battin's method had the longest runs times, at approximately 1,500 seconds, while the other three algorithms all had run time of approximately 840 seconds.

Table 2.4 Run times, in seconds, for each Lambert algorithm for each version of the grid search program.

	Serial Fortran	OpenMP	GPU
Battin	1,482.90	270.52	11.76
Gooding	837.74	159.59	8.40
Sun	845.93	164.93	7.85
Universal Variable	840.19	206.23	9.17

Table 2.5 Algorithm performance increases when compared to serial and parallel fortran

	Eff. over Serial Fortran		Eff. Compared to OpenMP
	OpenMP	GPU	GPU
Battin	5.48	126.14	23.01
Gooding	5.25	99.69	18.99
Sun	5.13	107.70	21.00
Universal Variable	4.07	91.65	22.50

Differences in the four solution algorithms become evident in both OpenMP and CUDA Fortran parallel implementations of the search program. For the OpenMP algorithms, Battin's method still produced the longest total run time, at 270 seconds. The universal variable method shows the least speed up with a run time of approximately 200 seconds. With the OpenMP parallel search, both Gooding's and Sun's methods have run times of approximately 160 seconds. This represents a relative increase in speed of approximately 25% increase when compared to the universal variable method. The Gooding and Sun algorithms converge within a very predictable number of iterations, typically no more than three and four iterations respectively, while the universal variable method can take anywhere between two to eight iterations. This large variation in the number of required iterations imposes some limitations on the speed increases possible with parallel algorithms. The low speed increase of the universal variable method is likely due to the large range of the number of required iterations.

The kernel developed for the CUDA search algorithm will suffer from some of the same synchronization issues as the OpenMP parallel algorithm. This effect is minimized by adjusting the block size and number of threads on which the kernel is executed on to ensure maximum performance of the algorithm. The final CUDA Fortran algorithms were able to process the entire 250 million Lambert solutions in just a few seconds. All four algorithms had closer

Table 2.6 Number of solutions per second, in millions of solutions, for each version of the Lambert algorithms.

	Serial Fortran	OpenMP	CUDA Fortran
Battin	0.17	0.92	21.27
Gooding	0.30	1.57	29.75
Sun	0.30	1.52	31.83
Universal Variable	0.30	1.21	27.27

performance on the GPU than on either CPU implementation. As with the CPU search implementations, Battin's algorithm had the longest average run times, at 11.76 seconds. While the Gooding algorithm was slightly faster than the Sun method on both CPU versions, it was approximately 6.5% slower on the GPU. With a run rime of 9.17 seconds, the universal variable method was 14% slower than the Sun algorithm, while the Battin algorithm was 33% slower.

A summary of each algorithm's relative performance increase for the three grid search programs can be found in Table 2.5. In general, the GPU search algorithms were approximately 100 times faster than standard Fortran, and 20 times faster than the parallel OpenMP versions. Battin's algorithm saw the largest performance increase when compared with both the serial and OpenMP at 126 and 23 times respectively. When compared with standard Fortran, the Gooding, Sun, and universal variable CUDA versions had approximate speed increases of 100, 107, and 92 times, respectively.

Ultimately, the most important performance aspect of any version of the algorithm is how many solutions per second can be obtained. This is true for any initial orbit determination or mission optimization algorithm. Table 2.6 shows the number of solutions per second, in millions of solutions, for all the algorithm configurations. On the GPU, Sun's method was the best algorithm, with a maximum of nearly 32 million solutions per second. Gooding' method was able to achieve approximately 30 million solutions per second, while the Battin and universal variable algorithms obtained 21 and 27 million solutions per second respectively. It should also be noted that the GPU results do not include the time it takes to transfer any data to and from the GPU. On the CPU, Gooding's method has the best performance at approximately 1.6 million solutions per second.

2.4 Concluding Remarks

The Battin, Gooding, and Sun algorithms are able to converge for nearly 100% of all orbit combinations. Both the Gooding and Sun algorithms have approximately the same performance, while the Battin algorithm is consistently slower. The universal variable method is conceptually the easiest method to understand and is the most commonly discussed method in the literature, but it lacks the robustness of the other three methods. If the efficiency of the algorithm is the most important factor, both Gooding's and Sun's methods, as implemented here, are likely the best to use.

The performance for each Lambert algorithm, when run on the GPU, has been compared. Each Lambert solution algorithm is able to compute tens of millions of solutions per second, with the Sun' method having the best performance at nearly 32 million solutions per second. This represents an increase in performance of two orders of magnitude when utilizing GPUs over standard CPU algorithms. While even the grid search CPU run times are not high when compared with common high performance computing algorithms, mission optimization algorithms often require run times ranging from hours to multiple days. By offloading computations for solutions to Lambert's problem to GPU(s) and sufficiently parallelizing optimization algorithms, performance increases of up to two orders of magnitude can be realized when compared with standard CPU algorithms. This will allow mission designers to quickly compute complex trajectories when evaluating potential mission architectures.

CHAPTER 3. ROBOTIC AND HUMAN EXPLORATION/DEFLECTION MISSION DESIGN FOR ASTEROID 99942 APOPHIS

In this chapter both robotic and human return missions to the asteroid Apophis are designed and analyzed. These mission are used to validate the solutions to Lambert's algorithm from the previous chapter, as well as other astrodynamics algorithms such as: solutions to Kepler's problems, date conversions, coordinate conversions, etc. The missions analysis and design performed in this chapter is done through and exhaustive grid search. The algorithms developed for this search are able to determine optimal launch dates and launch windows for simple rendezvous, direct intercept, and Earth return asteroid missions. These exhaustive search algorithms can then be used to evaluate the global convergence of trajectories found with the hybrid GNLP algorithm. This helps determine which options and algorithms should be used to determine optimal trajectories with the hybrid GNLP algorithm.

3.1 Introduction

Asteroids and comets have collided with the Earth in the past and will do so again in the future. Throughout Earth's history these collisions have had a significant role in shaping Earth's biological and geological histories. One major example of this is the extinction of the dinosaurs, which is widely believed to have been caused by the collision of an asteroid or comet. In recent years, near-Earth objects (NEOs) have also collided with the Earth, the most notable example in recent history is an impact in Siberia, known as the Tunguska event. This impact is estimated to have released an explosive energy of approximately $3-5$ megatons. While, the

impact occurred in a sparsely populated area, such an impact in a highly populated area would be extremely devastating.

Of all the NEO's found to date, the asteroid 99942 Apophis is considered one of the most potentially hazardous NEOs and has received much attention from the planetary defense community. However, an impact from Apophis does appear unlikely, with an estimated impact probability of approximately four-in-a-million in 2036. On April 13, 2029, Apophis will pass by the Earth, inside the limits of geostationary orbit. If Apophis passes through a relatively small, approximately 600-meter keyhole, impact will occur on April 13, 2036. A fictional scenario in which Apophis passes through a keyhole in 2029 and impacts with the Earth in 2036 is studied in this chapter. The purpose of this chapter is to perform the mission analysis and design for robotic and human exploration mission to Apophis, using the software algorithms developed for the Asteroid Deflection Research Center (ADRC). Possible launch windows, trajectories, and accompanying ΔV's for both robotic rendezvous and human piloted return missions prior to the April 13, 2029 Earth-Apophis close encounter will be analyzed. In addition, mission analysis and design will be performed for robotic and human piloted missions for the fictional scenario in which Apophis passes through a keyhole on April 13, 2029, resulting in an impact on April 13, 2036. The orbital current estimated orbital elements of Apophis, the fictional orbital elements, and the estimated physical characteristics can be found in Tables 3.1(a), 3.1(b), and 3.1(c), respectively [26]. For the fictional Apophis mission, launch windows will be determined throughout the 7-year period (keyhole passage through impact), which allow sufficient time for a fictional high-energy nuclear deflection mission.

A preliminary Interplanetary Ballistic Missile (IPBM) architecture, designed by the ADRC, will be used as the reference robotic space system throughout this chapter. The IPBM architecture is similar in design to the ADRC's hypervelocity asteroid intercept vehicle (HAIV). The most capable IPBM architecture, uses the Delta-IV Heavy launch vehicle, and is capable of a total ΔV of 4 km/s, carrying a 1500-kg nuclear payload. In addition, the reference departure orbit for the robotic mission analysis, used when determining the Earth-departure ΔV, is assumed to be a geostationary transfer orbit [27]. Using this baseline IPBM architecture

and ΔV capabilities, launch windows for both the pre-2029 and post-2029 missions have been determined in [28].

3.2 Human-Piloted Mission

To determine the feasibility of a human-piloted mission to Apophis, the mission requirements must first be determined. In particular, the minimum total ΔV necessary to complete the mission and the accompanying launch windows must be found. In this chapter the Lambert and other astrodynamic algorithms are used to determine optimal launch opportunities.

The algorithms are used to find the required ΔV's for each maneuver, find optimal launch windows, and to plot of resulting trajectories and other necessary information. A search is performed to determine the minimum ΔV for each launch date by performing and exhaustive search of all the possible Apophis arrival and departure date combinations, given only a desired mission length. An exhaustive search is used to ensure no minimums are missed. For the following analysis, a 185-km circular orbit is used for the departure parking orbit. To help minimize the total required ΔV, the atmospheric entry velocity is limited to a maximum of 12 km/s. Throughout this entire section an Apophis stay time of 10 days is assumed. Increasing or decreasing the stay time will result in a slight increase or decrease of the required ΔV. Results obtained for both the 180- and 365-day missions to Apophis near the 2029 close encounter, as well as the 2036 human-piloted deflection mission are presented and analyzed in this section.

Table 3.1 Orbital and physical parameters used for the real and hypothetical Earth impacting Apophis orbits.

(a) Current orbital elements

Elements	**Value**
Epoch	6/18/2009
a, AU	0.9224
e	0.1912
i, deg	3.3314
Ω, deg	204.443
ω, deg	125.404
θ_0, deg	134.713

(b) Fictional orbital elements

Elements	**Value**
epoch MJD	64699
a, AU	1.108243
e	0.190763
i, deg	2.166
Ω, deg	70.23
ω, deg	203.523
M_0, deg	227.857

(c) Physical parameters

Physical Parameter	**Value**
Rot. Per. (hr)	30.5
Mass (kg)	2.1E+10
Diameter (m)	270
H	19.7
Albedo	0.33

3.2.1 2028-2029 Launch Opportunities

For a human-piloted return mission to an asteroid, two possible launch windows are always found near the Earth-asteroid close encounter. One launch always returns to the Earth near the Earth-Apophis encounter date, while the other launch date occurs on the date of the close encounter. Throughout the rest of this chapter the mission prior to the Earth-Apophis close encounter will be referred to as the early launch date/window, while the launch occur refers to mission launch at or near the close encounter.

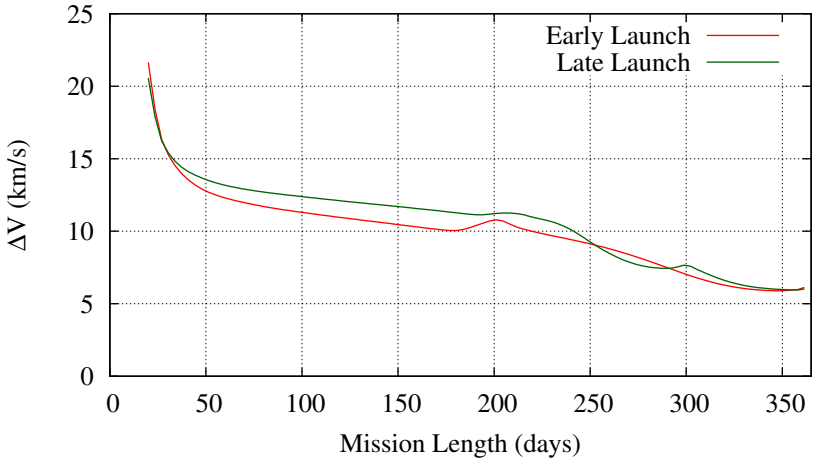


Figure 3.1 Plot of minimum ΔV required for the early and late launch as a function of the mission length.

A plot of the total ΔV required for both the early and late launch dates versus mission length (ranging from 20 to 365 days) is shown in Fig. 3.1[29]. As shown in Fig. 3.1, the total ΔV is, in general, reduced as the length of the mission increases. Fig. 3.1 shows that a local minimum for the required total ΔV occurs near the 180-day mission length. Current crewed NEO studies have limited the maximum mission length to 180 days for supply and maximum radiation dose limitations. Therefore, a complete mission analysis and launch window search for a 180-day mission length results in a required ΔV in the 10-11 km/s range. Lowering the total mission length may be possible depending on the mission architectures and ΔV capabilities.

3.2.1.1 180-Day mission analysis

With a total mission length selected, further analysis can be performed to find the dates and length of each launch window. This can be obtained by calculating the minimum launch ΔV for launch dates near the Earth-Apophis encounter. Launch opportunities can be found from Fig. 3.2, which is a plot of launch dates versus the total required ΔV for the selected 180-day mission. The first optimal launch date occurs approximately 180 days prior to the April 13, 2029 Apophis encounter, while the late launch date occurs during the asteroid close flyby on April 13, 2029.

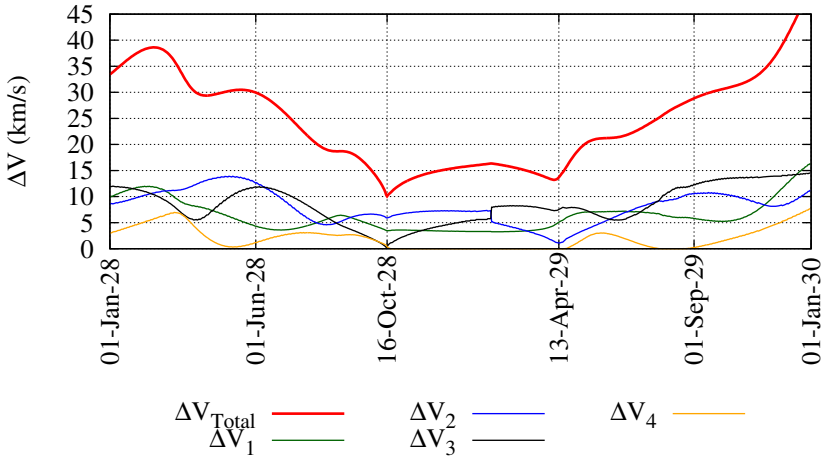


Figure 3.2 Launch date(2028-2038) versus minimum ΔV required for 180-day return mission.

Further examination of Fig. 3.2 reveals that the launch windows near the asteroid flyby are the last opportunities to launch a quick return mission to Apophis. Any human missions to Apophis after the April 13, 2029 launch date would likely require significantly increased mission times, possibly even multiple revolutions around the sun prior to rendezvous, to reduce the required ΔV to an obtainable amount. A mission of this length would likely require significant modifications to the Orion spacecraft to allow for greater radiation shielding and amount of supplies carried. For a short quick return mission to Apophis, the April 13, 2029 launch is the last easily obtainable launch date.

Limiting the maximum allowable launch ΔV to 11.5 km/s allows for sufficiently large launch windows. The minimum ΔV capability requirements for the mission are determined by allowing

for a 0.5-1 km/s error margin. Adding this error margin to the maximum allowable launch ΔV results in a required ΔV capability of 12-12.5 km/s. Using a 11.5 km/s limit, the launch windows can be found for both launch dates. Fig. 3.2 shows the total ΔV plot as well as the required ΔV for the Earth departure, Apophis arrival, Apophis departure, and Earth arrival burns. As Fig. 3.2 shows, the early launch window is approximately 12 days starting on October 12, 2028 and ending on October 24, 2028. The late launch window is significantly shorter at just over 2 days in length, ranging from April 12-14, 2029.

A summary of nominal launch dates for both launch opportunities can be found in Table 3.2. The dates for each maneuver as well as the ΔV magnitude and C_3 values are given for each maneuver. For the early launch date, all of the maneuvers, with the exception of the Earth departure burn, are carried out in the last 3 weeks of the mission. The return date of the early launch date mission is just after the Apr. 13, 2029 Earth-Apophis encounter, which allows for a small return ΔV because the Orion spacecraft departs Apophis a few days prior to the Earth-Apophis encounter. The opposite is true for the late launch date mission. Earth departure occurs during the Earth-Apophis close approach, with the Apophis rendezvous occurring a few days after Earth departure. Within the first 2-3 weeks, the mission is completed, with the remaining time spent on the return cruise. No burn is necessary when the Orion spacecraft returns to the Earth because the atmospheric reentry speed is less than 12 km/s.

3.2.1.2 365-Day mission analysis

From Fig. 3.1, it is clear that longer missions result in significantly lower ΔV requirements. It can be seen that extending the mission to 1 year in length results in a mission requiring a total ΔV of 6-7 km/s, significantly lower than the 10-12 km/s required for a 180-day mission. Such a mission could likely be executed using the previously proposed Constellation or similar mission architecture. This mission would likely be used as a stepping stone between the first asteroid mission, in the 2025 time frame, and the first Mars flyby mission, to occur in the 2035 time frame. Such a mission could serve as a test bed for systems needed for a Mars flyby or landing mission, with a reduced mission time when compared to a Mars mission.

A plot of the total ΔV needed for 2028-2029 time frame missions can be seen in Fig. 3.3. Fig. 3.3 shows the total ΔV needed for both the early and late launch date is just under 6.5 km/s, which is likely within the limits of Ares V/Ares I class of launch vehicle(s), carrying a CEV such as the Orion crewed capsule. Both launch windows can be determined by limiting the total ΔV to 7 km/s. The early launch window has a length of 27 days lasting from from April 12, 2028 to May 9, 2028. The second launch window is slightly longer at 35 days, ranging from March 11, 2029 to April 15, 2029.

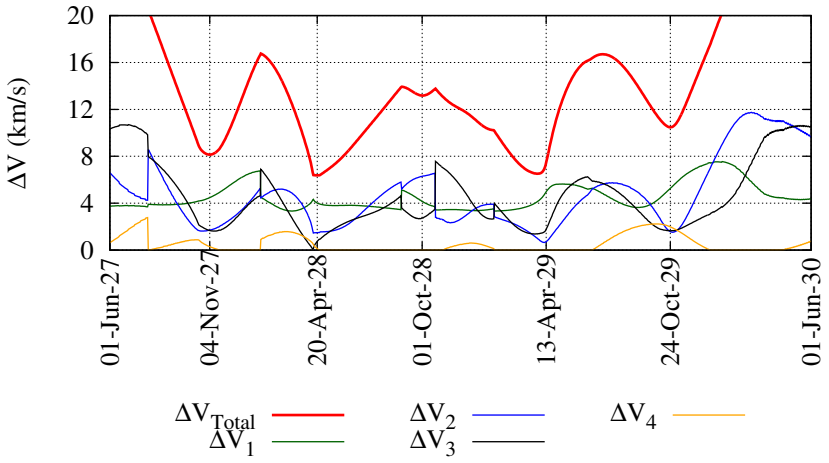


Figure 3.3 Launch date (2028-2029) versus minimum ΔV required for 365-day return mission.

A summary of the two nominal launch dates is shown in Table 3.2. For the early launch date, all of the maneuvers, with the exception of the Earth departure burn, are carried out in the last 2 months of the mission. The return date for the early launch date is just after the April 13, 2029 Earth-Apophis close encounter, which allows for a small return ΔV because the CEV departs Apophis a few days prior to the Earth-Apophis encounter. The opposite is true for the late launch date. Earth departure occurs during the Earth-Apophis close approach, with the Apophis rendezvous occurring a few days after Earth departure. Within the first $4-5$, weeks the mission is completed, with the remaining time spent on the return cruise. In each case a burn is necessary when the CEV returns to the Earth because of the required atmospheric reentry speed of 12 km/s or less.

3.2.2 Launch Opportunities Prior to the 2036 Impact

This section provides an outline of the fictional human mission requirements necessary for an Apophis deflection mission prior to impact on April 13, 2036. This was done by ensuring that the Apophis departure date occurs on or before the impact date. All figures and tables in this section represent this requirement. For this mission there is no equivalent launch date for what was referred to as the late launch.

A plot of minimum ΔV for mission lengths from 20 to 500 days is shown in Fig. 3.4. Examination of Fig. 3.4 reveals that there is a minimum required ΔV of just under 8.5 km/s. A 180-day mission requires a ΔV of over 14 km/s, which is not feasible given current space propulsion technology. The 12 km/s ΔV required for a 180-day mission in 2029 is very difficult to achieve and would likely involve the use of orbital transfer vehicles that currently don't exist. This would be inappropriate for an Apophis deflection mission, if the livelihood of so many people is at stake. For this reason, the only mission analysis performed in this section will be a 365-day mission, which is near the minimum total ΔV shown in Fig. 3.4.

From Fig. 3.5 it can be seen that there is only one minimum, which occurs approximately one year prior to impact of the fictional Apophis. This plot shows the ΔV values for each maneuver as well. Limiting the total ΔV to 9 km/s allows for a 2 week launch window starting on April 11, 2035 and ending on April 25, 2036. Depending on the specific architecture chosen this windows may be adjusted.

A complete summary of the nominal launch date can be found in Table 3.2. For this mission, the majority of the maneuvers, with the exception of the Earth departure burn, occur within that last 43 days of the mission. The Earth arrival date occurs just after the April 13, 2036 impact, assuming that the crewed deflection mission is failed or aborted. The entry velocity of the CEV is 12 km/s, the maximum allowed by the searching software, and may require a skip re-entry depending on CEV requirements. It may also be possible to perform a skip re-entry in order to eliminate the Earth arrival burn and bring the total required ΔV down to approximately 8 km/s.

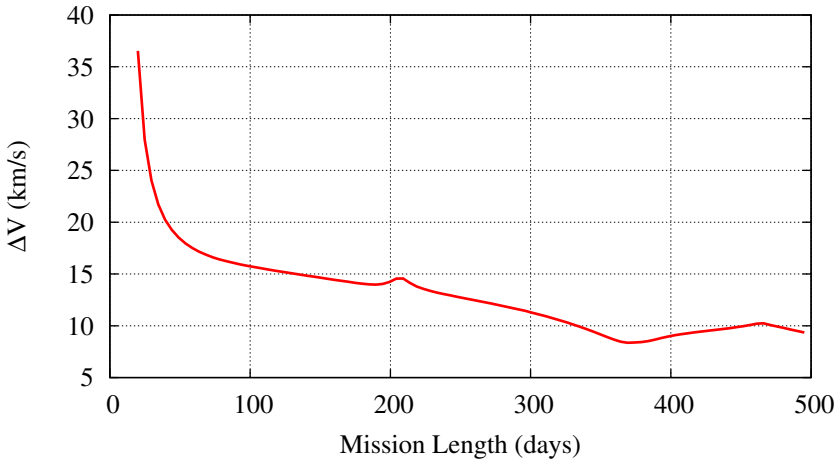


Figure 3.4 Plot of minimum ΔV required for the 2036 human-piloted human deflection mission.

3.3 Robotic Mission to Apophis

Prior to the execution of a human-piloted mission to Apophis, it may be necessary to have a robotic precursory mission. The objective of this mission would likely be to further refine the position of Apophis, send fuel/supplies prior to the manned mission, or for a robotic deflection mission. For this reason, mission analysis will be performed for Apophis up to the close encounter and for the fictional Apophis that will collide with the Earth in 2036. The mission analysis and design conducted in this section was performed using a similar computer program to the one used for human-piloted mission analysis.

3.3.1 Mission Analysis Prior to the 2029 Close Encounter

Launch opportunities were found by searching for the minimum total ΔV for each launch by allowing the arrival date to vary. The results of the search are shown in Fig. 3.6. Examinations of this plot shows several possible launch opportunities in the 2027-2029 date ranges, using the capabilities of the IPBM architectures. A summary of the launch opportunities can be found in [30]. A total of 6 launch windows were found, however only the first 4 launch windows allows for an arrival date before or during the manned mission. Additional launch windows could be found, if needed, by allowing multiple revolutions/phasing orbits prior to arrival at Apophis

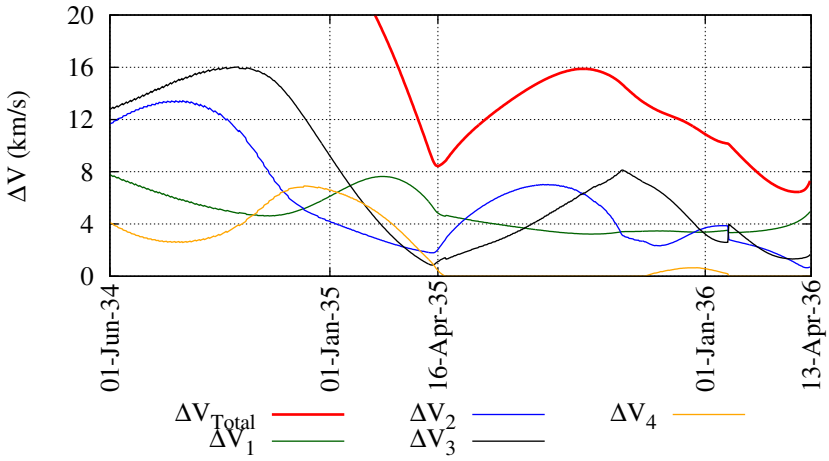


Figure 3.5 Launch windows found in the $2035 - 2036$ time frame.

or allowing a deep space correction maneuver. Additional information on a robotic mission to Apophis prior to 2029 can be found in [30].

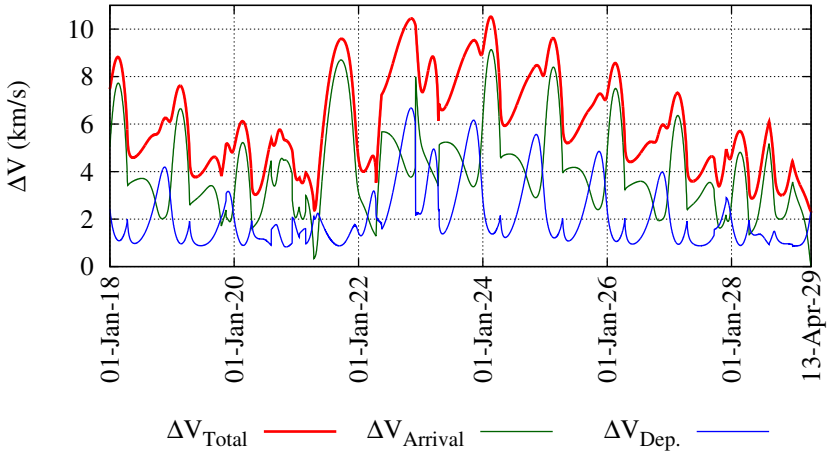


Figure 3.6 ΔV required for rendezvous mission.

3.3.2 Fictional Post-2029 Robotic Mission Analysis

A robotic mission after a close encounter, which results in an impact on April 13, 2036, would likely be a robotic deflection mission. Given this short period of time, missions other than a direct 0-revolution transfer may be necessary, to determine launch windows during times such missions don't allow. In this section, both 0-revolution and multiple revolutions missions

Table 3.2 Mission information for each launch opportunity.

	Early-180	**Late-180**	**Early-365**	**Late-365**	**2036**
Earth Departure					
Date	10/16/28	4/13/29	4/20/28	4/13/29	4/16/35
C_3	4.887	30.355	18.025	30.762	38.021
ΔV (km/s)	3.448	4.528	4.017	4.545	4.836
Apophis Arrival					
Date	3/26/29	4/19/29	2/21/29	5/11/29	3/3/36
V_∞^2	34.504	0.136	2.211	0.065	4.185
ΔV (km/s)	5.874	0.369	1.487	0.255	2.046
Apophis Departure					
Date	4/5/29	4/29/29	3/3/29	3/21/29	3/13/36
C_3	0.113	40.686	0.546	1.946	1.254
ΔV (km/s)	0.336	6.379	0.739	1.395	1.120
Earth Arrival					
Date	4/14/29	10/10/29	4/20/29	4/13/30	4/15/36
V_∞^2	30.474	1.896	24.053	21.269	31.569
ΔV (km/s)	0.391	0.000	0.129	0.014	0.435
Entry Vel. (180 km alt)	12.000	11.111	12.000	12.000	12.000
Total ΔV (km/s)	10.049	11.276	6.373	6.208	8.436

will be analyzed. It will also be shown that a direct intercept mission, in which no arrival burn is necessary, is possible during nearly the entire 7 year period. Additionally, recent studies have concluded that is may be feasible to significantly reduce the impact damage from an Earth-impacting NEO using a nuclear subsurface explosion as late as 15 days prior to impact [31].

3.3.2.1 0-Revolution mission analysis

A contour plot of the total ΔV for the time-of-flight (the number of days to rendezvous) versus launch dates is shown in Fig. 3.8. This kind of contour plot is often called the porkchop plot. Careful analysis of this porkchop plot indicates that launch windows will be available from 2029 into the early 2030's. Although hard to see, there is one short 5 day launch windows in 2035 as well. The main constraint for launch windows after 2035 is that interception must

occur at least 15 days prior to the Earth-Apophis collision [31, 32]. For this reason, all figures and tables in this chapter represent only missions which arrive at least 15 days prior to impact.

Launch windows can be determined by examining Fig. 3.7, which is a plot of minimum total ΔV versus launch date from 2029 to 2036. To determine the launch windows a maximum ΔV of 4 km/s was used, corresponding to the chosen IPBM configuration. There are 4 possible launch windows from 2029 to 2031 and a short 5 day launch window in April of 2035. Between 2031 and 2035 and after the 2035 launch window, there are no possible rendezvous launch windows prior to impact in 2036. Information for the nominal departure date for each windows can be found in Table 3.3. This table shows the magnitudes and dates for the Earth departure and Apophis arrival burns as well as the starting and ending dates for each corresponding launch window.

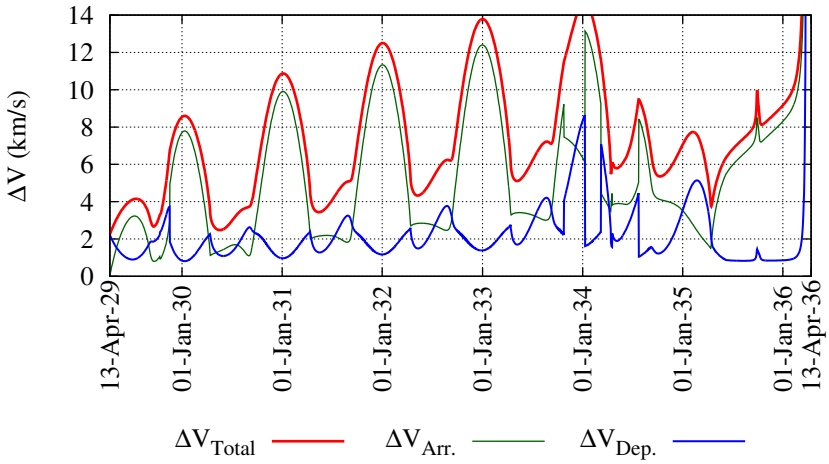


Figure 3.7 ΔV required for rendezvous mission from 4/13/2029 to 4/13/2036.

3.3.2.2 0-Revolution direct intercept mission

After April of 2035, there are no feasible rendezvous launch windows, which means that a direct intercept mission would be required for any "last minute" disruption missions. It is likely that developing and building a spacecraft such as the proposed IPBM system would take several years. It also seems likely that development would not start until after Apophis has passed through a keyhole in 2029, this is due to the four-in-a million chance this will happen.

Table 3.3 Minimum ΔV transfer trajectory for each optimal 0-revolution launch date. ΔV's are given in km/s.

Launch Window	Dep. Date	Dep. ΔV	Arrival Date	Arrival ΔV	Total ΔV	Start	End	Days
1	4/13/29	2.186	4/16/30	0.102	**2.289**	4/13/29	6/23/29	**71**
2	9/12/29	1.867	7/26/30	0.801	**2.668**	8/12/29	10/24/29	**73**
3	5/20/30	1.091	6/2/31	1.460	**2.552**	4/11/30	9/12/30	**154**
4	5/15/31	1.281	8/30/32	2.156	**3.436**	4/20/31	6/28/31	**69**
5	4/15/35	2.045	3/28/36	1.702	**3.747**	4/13/35	4/18/35	**5**

With the last low ΔV rendezvous 0-revolution launch window ending a little more than 2 years after the 2029 close encounter, it may be desirable to launch either a direct intercept or multiple revolution mission. For this reason, a direct intercept and multiple revolution missions will be analyzed in the following sections. A total ΔV contour plot of the time-of-flight versus launch date is shown in Fig. 3.9. From this plot, it can be seen that an intercept mission with a maximum ΔV of 4 km/s is possible at almost continuously from 2029-2036.

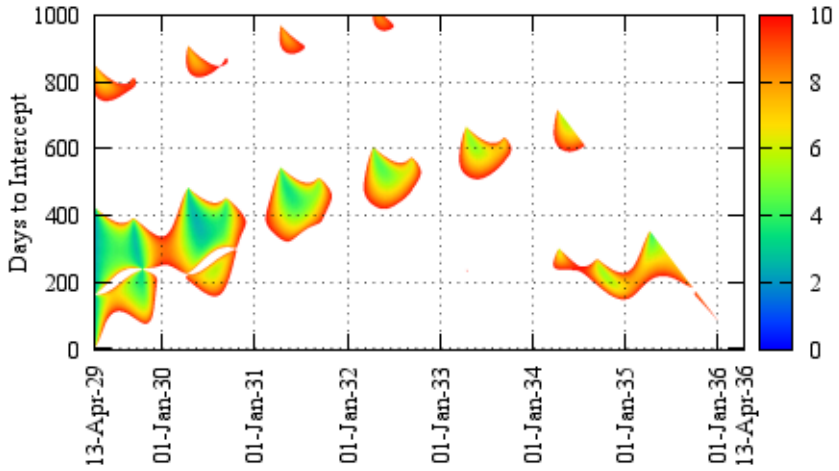


Figure 3.8 Total ΔV porkchop plot of the time-of-flight versus launch date for the rendezvous mission.

Recent research [31, 32] has shown that it may be possible to prevent all but a few percent of the material from an asteroid from impacting the Earth by utilizing a nuclear subsurface explosion. However, such a disruption mission requires a rendezvous with Apophis to provide

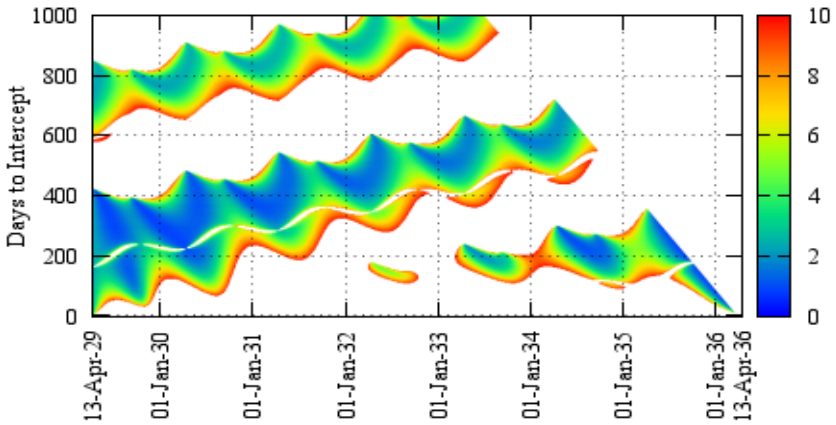


Figure 3.9 Porkchop plot for the direct intercept mission.

an acceptable penetrator velocity of 300 m/s. However, it may also be possible to design a high-speed penetrator. The objective of this section is to determine the feasibility of a last minute rendezvous mission launched anywhere from 2-3 years to as little as $20 - 30$ days prior the the 2036 impact. This requires further analysis of Fig. 3.9. An expanded version of Fig. 3.9 showing only this time span is provided in Fig. 3.10.

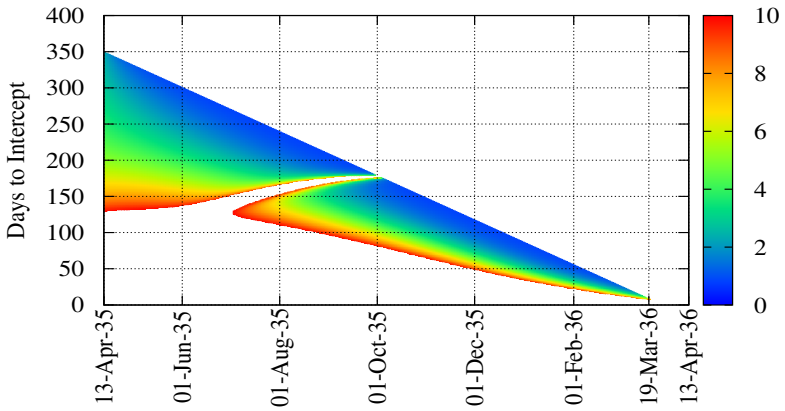


Figure 3.10 Total ΔV contour plot of the flight-of-time versus launch dates from 4/13/2029 to 4/13/2036 for the intercept mission.

Careful analysis of Fig. 3.10 shows that late launch dates ranging from 3/28/2035 to 3/19/2036 are feasible for the intercept mission. The last feasible launch date is 25 days prior to impact. Using the IPBM system architecture, it may be feasible to launch a last minute

intercept mission to disrupt an NEO similar to Apophis. No limits on the arrival velocity have been imposed in Figs. 3.9 and 3.10. Unfortunately, the penetrator's maximum impact velocity is currently limited by 300 m/s, which means that the late intercept mission concept may not be a viable option. In this situation either a nuclear surface explosion (contact burst) or high-speed penetrator (>5 km/sec impact velocity) must be developed and employed. An example of late intercept trajectory is shown in Fig. 3.11, with all necessary mission data shown in Table 3.4.

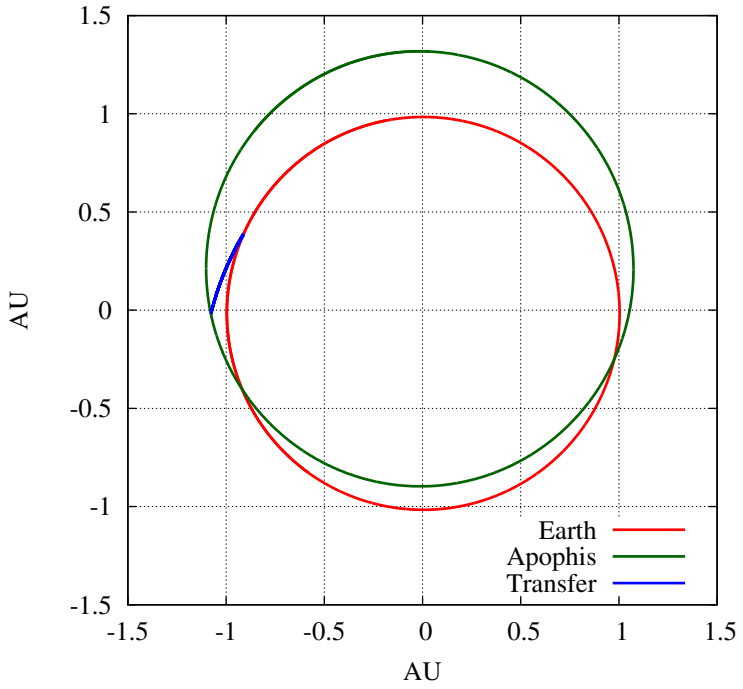


Figure 3.11 Trajectory for a late intercept mission. Trajectory shown in (X,Y)-plane of J2000 coordinate system.

To fully determine the feasibility of the late intercept mission, further information is needed on an intercept penetrator used for the nuclear subsurface explosion. In general, the later the intercept launch occurs the higher the arrival V_∞ at Apophis. Intercept launch dates in the 2034-2035 range generally have an intercept velocity (at Apophis) in the 5-6 km/s range.

Table 3.4 Summary of a last minute intercept mission

Example Trajectory	
Departure Date	26-Feb-36
Arrival Date	19-Mar-36
Total ΔV, km/s	2.830
Semi-major axis, AU	1.714
Eccentricity	0.441
Inclination, deg	2.032
Ω, deg	336.944
ω, deg	153.860
Departure θ, deg	26.163
Arrival θ, deg	50.231

3.3.2.3 Multiple revolution rendezvous mission analysis

For a search for multiple revolution missions two solutions are always found. A low-energy and high-energy solution. These indicate two solutions with a small and large eccentricity respectively. As implied by the name, ΔV's required for the high-energy solution are almost always larger than the corresponding low energy solution. Consequently, most of the additional launch dates determined by the multi-revolution (prior to Apophis arrival) search are part of the low-energy solutions. For this reason, only the low-energy missions will be analyzed in this section.

A porkchop plot of the number of days to Apophis interception versus launch date, similar to Fig. 3.9, is shown in Fig. 3.12. From this porkchop plot, it can be determined that the optimal mission lengths occur in the 600-1000 days range, slowly increasing as the launch date increases. Several launch dates similar to the 0-revolution rendezvous mission exist prior to May of 2031. However, it can be easily seen from the porkchop plot that additional launch windows exist as well during other unaccessible period for the 0-revolution mission. These additional launch windows occur from April of 2031 to April of 2034, a period where rendezvous 0-revolution missions were not possible.

The 8 possible launch windows can be easily seen in Fig. 3.13. This is a plot of the total ΔV as well as the Earth departure and Apophis arrival ΔV's. For all dates searched, the last

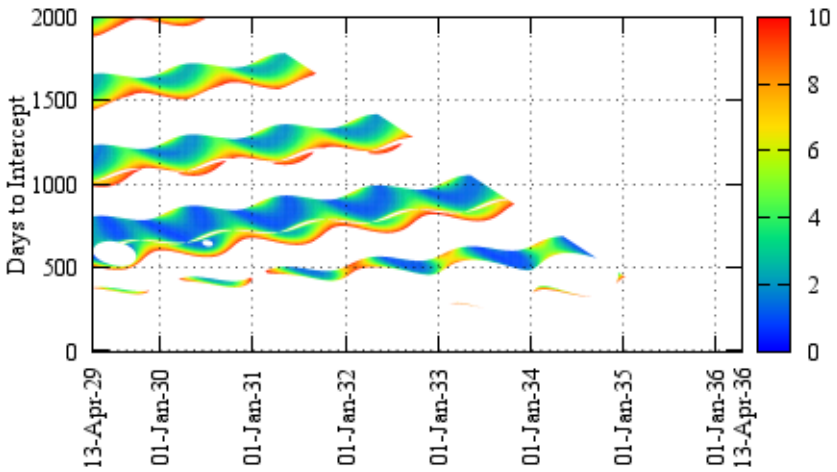


Figure 3.12 Total ΔV contour plot of the time-of-flight versus launch dates from 4/13/2029 to 4/13/2036 for the 1-revolution low-energy solution rendezvous mission.

possible arrival date is March 29, 2036, 15 days prior to impact. The intercept date limitation is the reason that no 1-revolution solutions exist after early 2035, particularly low ΔV solutions. Analysis of this plot shows that a total of 8 launch windows exist, with the last 5 windows opening up launch dates that were previously unavailable for the 0-revolution rendezvous mission. The accompanying window ranges and information for the nominal departure date for launch window are shown in Table 3.5. In general the launch window lengths are similar, perhaps slightly longer for the 1-revolution case when compared to the 0-revolution windows. However, when performing a $2-3$ revolution search, the early launch windows are drastically lengthened when compared to the 0-revolution case. No additional launch windows were found for 2+ revolution case, so no further analysis will be presented.

3.4 Summary

The mission requirements for the $2028-2029$ human-piloted exploration mission have been discussed in this chapter. It has been shown that a 180-day mission, currently the longest considered in many NASA studies [33–35], requires a ΔV of approximately 12 km/s. However, it has been shown that the ΔV requirements can be significantly lowered by extending the mission length to 1 year. In this case the required ΔV is in the 6-7 km/s range. This amount

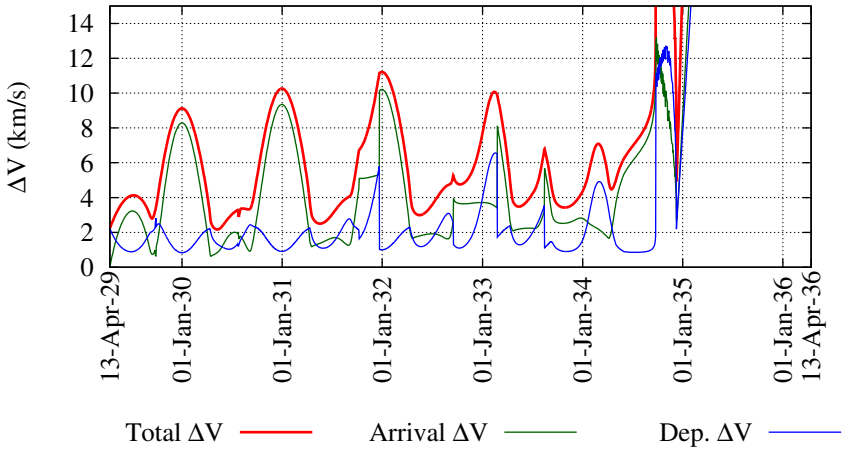


Figure 3.13 ΔV required for rendezvous mission from 4/13/2029 to 4/13/2036 for the 1-revolution low-energy solutions.

ΔV can be obtained with a mission architecture similar to the previously proposed Constellation systems carrying a crew exploration vehicle such as the proposed Orion capsule[30]. A mission of this length could be used as a stepping stone between the first human NEO mission and a Mars mission.

The requirement for the fictional Apophis human-piloted deflection has been determined as well. In this situation, only the early launch windows, which occurs prior to impact with an arrival occurring approximately 1 month prior to impact. In such a situation a high-energy nuclear method would likely have to be employed as the deflection/disruption technique. Similarly to the 2028-2029 exploration mission, the minimum ΔV occurs with a mission length of approximately 1 year. The required Δ for this deflection mission is higher than many proposed exploration mission, at approximately 8.5 km/s. [30].

Mission analysis has also been performed for robotic exploration and deflection missions prior to the 2029 close encounter and for the fictional orbit resulting in impact on April 13, 2036. A total of 6 launch windows, with 4 arriving prior to the April 13, 2029 close encounter. For the post-2029 fictional, Earth-impacting Apophis orbit, 5 launch windows were found for the 0-revolution mission. Additional launch windows were found utilizing a 1-revolution Lambert solution, resulting in eight possible launch windows, with many of these occurring when 0-revolution rendezvous missions weren't possible. It has also been shown that a 0-revolution

direct intercept mission is possible, depending on the final arrival velocity limits, for nearly the entire 7-year period.

3.5 Conclusion

Asteroid 99942 Apophis can be an excellent target for human exploration of NEOs. With a mission length of 1 year it could be used to learn valuable information about extended deep space missions, while remaining close to the Earth. This mission could prove invaluable for later Mars missions. In the unlikely event that Apophis passes through a keyhole in 2029, a deflection mission, such as those outlined in this chapter, would become necessary. Research currently being done at the ADRC indicates that such a mission may be feasible using current technology and launch vehicles.

Table 3.5 Minimum ΔV transfer trajectory for each optimal 1-revolution low-energy launch window, ΔV's are given in km/s.

Launch Window	**Dep. Date**	**Dep. ΔV**	**Arrival Date**	**Arrival ΔV**	**Total ΔV**	**Start Date**	**End Date**	**Days**
1	4/13/29	2.242	5/7/31	0.013	**2.255**	4/13/29	6/16/29	**64**
2	9/24/29	2.053	9/25/31	0.7394	**2.793**	7/30/29	10/4/29	**66**
3	5/10/30	1.324	6/29/32	0.8449	**2.169**	4/5/30	9/18/30	**166**
4	5/20/31	1.123	10/3/33	1.3812	**2.505**	4/11/31	7/27/31	**138**
5	5/16/32	1.194	12/22/34	1.7989	**2.992**	4/13/32	7/18/32	**96**
6	5/13/33	1.293	3/2/36	2.1871	**3.480**	4/18/33	6/22/33	**65**
7	10/28/33	0.905	5/4/35	2.5233	**3.429**	9/18/33	12/17/33	**90**
8	4/15/34	2.008	3/28/36	1.7413	**3.750**	4/12/34	4/18/34	**6**

CHAPTER 4. DEVELOPMENT OF THE HYBRID OPTIMIZATION ALGORITHM

This chapter discusses the selection of global optimization algorithms and the development of the hybrid genetic-nonlinear programming (GNLP) algorithm. This algorithm has been designed for the purposed of optimizing interplanetary trajectories, including multiple gravity-assist (including the possibility of adding in deep space maneuver(s)) and low-thrust trajectories, but can be used for a variety of optimization problems. The performance of the GNLP algorithm will be evaluated on several optimization test functions prior to being used for the interplanetary trajectory optimization presented in the following chapters.

4.1 Introduction

Several optimization algorithms have been developed to optimize mission design problems, particularly for the MGA and MGA-DSM problems [36–43]. Typical optimization algorithms developed for these types of missions often used nested evolutionary algorithm. A genetic algorithm is often used for the outer loop, which optimizes the flyby order, while separate algorithm(s) are used to optimize the variables for individual trajectories. This inner loop mission trajectory is often optimized through the use of a cooperative algorithm, which typically alternate between particle swarm, differential evolution, or genetic algorithms [39, 40]. Other research groups have focused on using advanced genetic algorithms as their primary optimization algorithm [36–38]. Because these methods don't take advantage of gradient based optimization, these types of algorithms often have long run times (typically 1+ days) and frequently only find near optimal trajectories.

The motivation behind the development of the hybrid optimization algorithm is to develop an algorithm than can significantly reduce the total computational cost of the optimization process, while autonomously determining optimal trajectories for both impulsive (MGA and MGA-DSM missions) and low-thrust trajectories. This is done by combining the global converge characteristics of evolutionary algorithms with the local convergence of traditional nonlinear programming gradient based optimization methods.

4.1.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are stochastic search algorithms which are often used for global optimization. These algorithms are designed to emulate the natural processes of biological evolution. They typically operate on a population, consisting of potential problem solutions, and use genetic principles, such as survival of the fittest, crossover, mutation, and reproduction to search for more optimal solutions. The processes employed by EAs essentially evolves the population to find more fit solutions. Different evolutionary algorithms have been utilized for a variety of mission design problems [36–38, 40–43]. Some of the most commonly used EAs are discussed below in an effort to help determine which evolutionary algorithm would be used with the hybrid optimization algorithm.

4.1.1.1 Genetic algorithm

The genetic algorithm has often been applied to mission design problems [36–38, 40], and is one of the first EAs to be developed. The problem decision vector is encoded in a set of real valued and integer sets, known as chromosomes. This is preferred over traditional binary encoding, because real valued crossover operators often increase the probability that crossover operators will result in a better solution [42]. Genetic operators, such as mutation, and reproduction are used to evolve the population towards more optimal solutions. The genetic algorithm was chosen as the global optimization algorithms for the hybrid optimization algorithm, further details can be found in the following sections.

4.1.1.2 Particle swarm optimization

Particle swarm optimization is a class of evolutionary algorithm, developed in the 1990s, that was originally design to simulate the flocking behavior found in birds and schools of fish [44]. Like genetic algorithms, each member of the population is randomly initiated, which is known as the solution's position. For these algorithms each member of the population is also assigned a randomly initialized velocity. The particles are then "flown" through the solution space to determine optimal solutions.

Particle swarm optimization algorithms keep track of the global best solution achieved and the best solution of the current generation. At each generation the velocity is changed by attracting the individuals in the population towards the current best solution and the global best solution. Various method to control the acceleration of the particles exist as well as modification to increase the effectiveness of these algorithms. One of the most common modifications include multiple particle swarm optimization, which evolves multiple populations and swaps particles from the separate swarms each few generations.

Particle swarm optimization has been utilized for mission optimization, but it is often used on conjunction with another type of evolutionary algorithm such as a genetic or differential algorithms [40]. Particle swarm optimization was considered for the mission optimization algorithm. However, the hybrid genetic algorithm, as discussed in this chapter, performed better for the mission optimization problems considered in this dissertation.

4.1.1.3 Differential evolution

Differential evolutionary algorithms are a class of global evolutionary algorithms, closely related to genetic algorithms, that were also developed in the 1990s [45]. There are numerous implementations of differential algorithm, but the differential algorithm considered in this study uses an implementation similar to the algorithm utilized by other research groups for spacecraft trajectory optimization [39, 40].

Like genetic algorithms, differential algorithms simulate the natural processes of mutation, crossover and selection. For the algorithm considered for use with the hybrid algorithm a

difference vector is created from four randomly selected members of the population. This difference vector is then used with the crossover operator. With differential algorithms the variables bounds are not enforced by the evolutionary operators, so the algorithm must enforce the variable bounds. During testing, the hybrid algorithm performed similarly when using either the genetic algorithm or the differential evolutionary algorithm. However, with the hybrid algorithm implementation presented in this chapter the differential evolution algorithm typically required a higher total number of function evaluations.

4.1.2 Simulated Annealing

Simulated annealing is a stochastic search method for global optimization [46]. The name comes from the annealing process used in metallurgy, which is a controlled process used to reduce defects in crystals. At high temperatures the molecules can freely move about, if the cooling process is done in a slow controlled method low energy organized crystal lattice is formed. This is the process that is reproduced with simulated annealing and adaptive simulating annealing algorithms.

The simulated annealing process is an iterative procedure that updates a single candidate until termination criteria are met [47]. The solution is randomly initiated and moved through the solution space by parameters set for the algorithm. This process has been applied to many optimization problems, including mission optimization problems [48]. However, they have not been used as widely used as evolutionary algorithms [36–38, 40–43]. Because of their complexity a simulated annealing algorithm was not used ultimately not use with the final hybrid optimization algorithm.

4.1.3 Local Optimization Methods

The most common local optimization algorithms employ newton or quasi-newton deterministic optimization methods. These method use an iterative procedure to improve the objective function with each iteration by utilizing information from both first and second order derivative to pick the direction of the search.

Newton method's require the gradient, Jacobian, and Hessian matrices, while the quasi-newton methods, often utilized in the nonlinear programming (NLP) algorithms rely on an approximation to the Hessian (2^{nd} derivative) known as a Broyden-Fletcher-Goldfarb-Shanno (BFGS) approximation. This is useful when the numerical calculation of the Hessian matrix is computationally expensive. Further details of the NLP algorithms are beyond the scope of this dissertation, but can be found in the references for each of the NLP algorithms implemented in the hybrid algorithm.

4.2 Development of the Hybrid Genetic-Nonlinear Programming Algorithm

Table 4.1 Summary of the numerical algorithms considered when developing the hybrid optimization algorithm.

	Evolutionary Algorithms	**Simulated Annealing**	**Quasi-Newton**	**Newton**
Convergence Radius	⋆⋆⋆	⋆⋆⋆	⋆	⋆
Convergence Rate	⋆	⋆	⋆⋆	⋆⋆⋆
Accuracy	⋆	⋆	⋆⋆⋆	⋆⋆⋆
Type	Stochastic	Stochastic	Deterministic	Deterministic

A summary of the optimization types considered for the potential hybrid algorithm are shown in Table 4.1. The motivation for the hybrid optimization algorithm is to develop an algorithm that is capable of solving complex mission design problems with no prior knowledge of the solution structure or approximate solution neighborhood, as is required for purely gradient based non-linear programming optimization. To accomplish this, a hybrid algorithm has been developed that combines the global optimization capabilities of evolutionary algorithms with the local optimization capabilities of traditional gradient based NLP solvers.

After preliminary testing a genetic algorithm was chosen as the global optimization algorithm. A total of three separate nonlinear programming (local optimization) solvers have been implemented to determine locally optimal solutions. Each of these solvers can be utilized and adjusted for specific problems. In addition, by utilizing local optimization algorithms, the genetic algorithm can now used for constrained optimization problems.

It will be shown that the final algorithm is capable of determining optimal(or at least very near optimal) solutions for preliminary low-thrust trajectories. The low-thrust trajectory optimization problem is known to be one of most difficult optimization problems in astrodynamics and is an area of active search [41, 48–52]. The hybrid GNLP algorithm is able to find complete solutions, including the number of gravity-assists and flyby order, for the two and three-impulse (MGA and MGA-DSM) class of problems.

4.2.1 Development of the Real Valued Genetic Algorithm

The heart of a genetic algorithm is the stochastic simulation of natural selection, reproduction, and mutations found in natural evolution. In these simulations, genetic operators are used to 'evolve' an initial population, through genetic operators, to determine the best fitness design [41]. The purpose of this section is to discuss the basics of the genetic algorithm developed for the final hybrid optimization algorithm. The genetic algorithm is responsible for the global optimization capabilities of the final algorithm.

A genetic algorithm is a stochastic optimization method based on the principles of evolution. Genetic algorithms perform a probabilistic search by evolving a randomly chosen initial population. The population is just a series of sets of variables that are evaluated by the objective function, which was developed in the previous section. The advantage of using evolutionary methods over traditional optimization methods is that no initial solution is necessary. This helps ensure, but does not guarantee, that solutions are not confined to a single locally optimal solution. Genetic algorithms also perform well in complex nonlinear and discontinuous design spaces. Despite all the advantages, evolutionary algorithms also have a downside. They almost always require a greater number of cost function evaluations than traditional gradient based methods, increasing the computational requirements. Additionally, evolutionary algorithms do not make use of gradients, so there is no proof of convergence. It should also be noted that genetic algorithms were developed for bound-constrained minimization problems and may not always perform well for unconstrained minimization when very large bounds are used. Many shortcoming of evolutionary algorithms are alleviated in the final hybrid algorithm. By uti-

lizing a local NLP gradient based solver the hybrid algorithm requires significantly smaller populations and will have provide at least locally optimal solutions.

The basic genetic operators include the following: selection, reproduction, crossover, mutation, and elitism. The algorithm developed in this section can utilize a number of different user selected selection, reproduction, crossover, and mutation methods. In a real valued genetic algorithm (as implemented for this study), each variable is represented by either its integer or real number value. Each real and integer variable is referred to as a gene. A complete set of variables, which defines a solution to the problem, forms a complete chromosome. A single chromosome then corresponds to one member of the entire population, which can contain hundreds to thousands of chromosomes. This real valued approach has an advantage over the traditional binary representation of variables because the real valued variables do not have a discrete resolution. This is especially important for problems that are sensitive to small changes in the variables. For the pure genetic algorithm, individual input variables include upper and lower bounds, size of the population, and various other parameters to control the flow and output of the final optimization routine.

The first step in the evolutionary process is to use uniform random numbers to generate the initial population. This is done in a way that ensures a completely random, evenly distributed initial population is used to start the problem. Each individual chromosome is generated using the integer and real valued user supplied upper and lower bounds. The process is then repeated until the entire population has been filled.

From this point each member of the population is assigned a fitness value via a user supplied objective function. The genetic operators are then used to generate a new population from the initial parent population. These genetic operators are crucial to the performance of the genetic algorithms because they enable a more fit population to be evolved from the initial population. This process continues until the genetic algorithm exit condition occurs, at which point the final population is return. Common stopping conditions include limiting the number of generations, monitoring when the best fit solution stop changing, or monitoring when the average cost function value approaches the population minimum. For this study the optimization algorithm is run out until the best solutions hasn't changed for at least 25 generations. The sequence

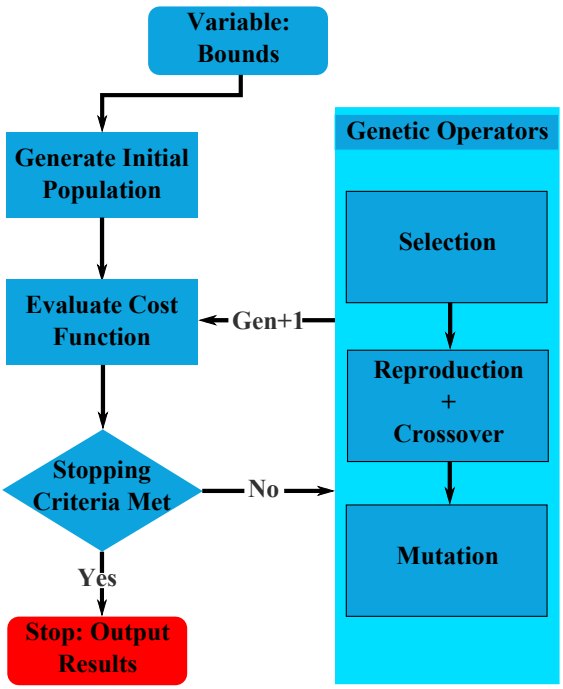


Figure 4.1 Flow chart for the simple real and integer valued genetic algorithm.

of operations of the core operations of the genetic algorithm is illustrated in Fig. 4.1. One secondary operator, elitism, is also required for the genetic algorithm. This operator simply ensures that the best fit solution(s) are not lost from generation to generations by directly inserting the best members of the parent population into the next generation.

4.2.1.1 Selection types

In genetic algorithms a selection operator is used to determine which individuals in the parent population pass on their genes to the next population, through the reproduction and crossover operators. This operator is representative of the survival of the fittest evolutionary principle. In this algorithm, a total of two selection types have been implemented, one based solely on a roulette selection method, and another that combines roulette and tournament selection.

Roulette selection is a fitness proportionate selection method in which better fit individuals are more likely to be chosen. In this selection method, a normalized fitness that ranges from 0 to 1 is used. The fitness is reformulated so that more fit members of the population have a higher normalized fitness value and will be more likely to be chosen to become parents.

The first step in this process is to evaluate the objective function for each member of the population. The value from the objective function evaluation is known as raw fitness, $r(i)$. In this case i represents the chromosome/population number. The raw fitness is then standardized, depending on whether the problem is being minimized and maximized. If the problem is a minimization problem, the standardized fitness is the same as the raw fitness, represented by

$$s(i) = r(i) \tag{4.1}$$

However, if the problem is a maximization problem, the standardized fitness is the individual raw fitness value subtracted from the largest raw fitness value. This maximization problem standardized fitness is represented by

$$s(i) = r_{max} - r(i) \tag{4.2}$$

Next the standardized fitness is adjusted so that each individual has a value between 0 and 1, with larger adjusted fitness values representing better fit individuals. The advantage of this step is that as the population matures, fitness values are exaggerated in population members that are close to the most fit solution. However, this adjustment is most effective for problems where the optimal solution is near 0 [53, 54]. To ensure the adjust fitness is between 0 and 1 the adjusted fitness is calculated as follows:

$$a(i) = \frac{1}{1 + s(i)} \tag{4.3}$$

Now that the adjusted fitness values lie between 0 and 1 the next step is to normalize the whole population, so the sum of the normalized fitness values is 1. This is done so that selected individual can be chosen from pseudo random number, which also have a range from 0 to 1.

Individual normalized fitness values are determined by dividing the individual adjusted fitness value by the sum of adjusted fitness values, which is express as

$$n(i) = \frac{a(i)}{\sum_{j}^{n} a(j)} \tag{4.4}$$

The actual selection of individuals is performed as follows. First the normalized fitness values are sorted from largest to smallest. A random number is then used to decide which individual is selected. The selection of an individual is done by a summing the normalized fitness values. The individual that causes the summation value to be greater than the random number is chosen to be a parent and move on to the next set genetic operators. This process is repeated until enough parents have been selected to maintain the same population size as the parent generation. Roulette selection ensures that the best fit individuals have the highest probability of surviving to the next generation.

The second selection method implemented is tournament selection. This method is a greedy over selection method which uses roulette selection in conjunction with standard tournament selection. In this case, greedy over selection methods means the algorithm performs additional measures to try to evolve to an optimal solution more quickly than the standard selection method. Two individuals are chosen, using roulette selection, to compete to determine which individual is allowed to pass on its genetic material. The winner of the competition is done by directly comparing the normalized fitness values. The most fit individual is then selected to become a parent. This process is continued until enough parents are chosen to fill the next population.

This selection method has an advantage because it drives down the raw fitness and average of the fitness function in fewer generations than pure roulette selection. As with all greedy over selection methods, there is a risk that genetic diversity, which could help the population in later generations, will be lost. Although tournament selection will likely result in faster convergence, it should be used with care. For both the MGA (multiple gravity-assist) and MGA-DSM (deep space maneuver) problems, both selection methods work well.

4.2.1.2 Reproduction and crossover operator types

While the selection operator determines which population members get the privilege of reproducing and passing on their genetic material to future generations, the crossover and reproduction operators decided what is done with the genetic material. The user must supply the probability than an individual will under go either reproduction or crossover. To maintain a constant population size, these two probabilities must total up to 1.0 (100%). Crossover and reproduction values should typically be chosen close to 0.9 and 0.1, respectively. Both of these operators require two parent and produce two offspring.

The simplest of these two operators is reproduction. Reproduction occurs when two parents are passed directly from the parent generation into the next generation. As with every other critical procedure in the genetic algorithm, whether the parents undergo reproduction or crossover is chosen by a random process using the user supplied probabilities. In the implemented genetic algorithm a random number is generated, if the random number is greater than the user given crossover probability the parents undergoes reproduction. If not, a crossover operation is performed. Without the reproduction and elitism operators genetic algorithms would be no more useful than random searches.

The crossover operator, which represents biological reproduction, is used to allow individuals to be created with new and unique genetics. The purpose of each crossover type is to promote genetic diversity and expand, in a controlled way, the search of the design space. Unlike the reproduction process, crossovers allow new points in the design space to be searched. The crossover operators produce two offspring that contain genetic material from both parents. A total of five distinct crossover types have been implemented for use with the hybrid GNLP algorithm. The crossover type is selected by the user, making the final hybrid algorithm suitable for many different optimization problems. For typical mission design problems either the double point or uniform crossover operators produce the best results.

Single Point Crossover The first crossover operator is single point crossover. In the single point crossover operator one variable is randomly chosen to be the crossover point. The first child is created by copying everything from the first parent prior to the crossover point and

Figure 4.2 Illustration of the single point crossover with the 3rd variable chosen as the crossover point.

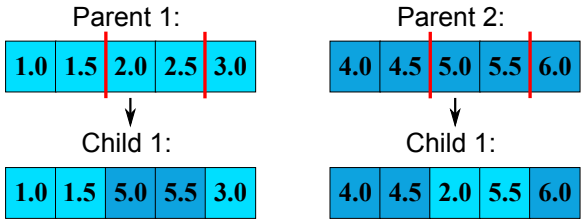


Figure 4.3 Illustration of the double point crossover with the 3rd and 5th variables chosen as the crossover point.

everything from the second parent after the crossover point. The second child is the inverse of this process. This process is shown graphically in Fig. 4.2.

Double Point Crossover Double point crossover is similar to the single point crossover operator. In this case two crossover points are randomly chosen. After the two points are chosen two new individuals are formed, as shown in Fig. 4.3, by swapping the variables between the two crossover points.

Uniform Crossover Uniform crossover has been shown to be a very effective method for promoting genetic diversity, and, in turn, discovering new useful chromosomes [41, 55]. With the uniform crossover operator each variable in the chromosome is a crossover point. The two offspring are then generated by a series of virtual coin flips. The virtual coin flip is used to decide which offspring gets the genetic material from the separate individual parents. While this process promotes genetic diversity, it is best used to determine optimal solutions near an approximate solution that has been previously determined. This can be done by using this operator late in the search, when genetic diversity have begun to stagnate or my running a

Figure 4.4 Illustration of the arithmetic crossover operator with $\alpha = 0.25$.

smaller search localized around a sub-optimal point. Graphically this process is similar to both the single and double point crossover methods.

Arithmetic The arithmetic crossover operator, sometimes referred to as the whole arithmetic operator, linearly combines the chromosomes from the two parents. The two offspring are generated from is by linearly combining the two parents with a randomly chosen multiplier, α, which has the range of 0 and 1. It is possible to use other bounds for α, as long as the upper and lower bounds are checked for each resulting gene. The two offspring are determined from the random multiplier as follows:

$$C_1 = \alpha P_1 + (1 - \alpha) P_2 \tag{4.5}$$

$$C_2 = (1 - \alpha) P_1 + \alpha P_2 \tag{4.6}$$

A simple illustration of the arithmetic operator can be seen in Fig. 4.4. This illustration shows a chromosome consisting of only real valued variables, but the method can be easily extended to include integer value variables by rounding to the nearest integer.

Heuristic The heuristic crossover operator determines a search direction from the two parent members. This is similar to the arithmetic operator, but modifies the search direction from by moving the worst solution towards the better parent. The two offspring are created using a randomly chosen variable, r, which has a range from 0 to 1, as outline below

$$C_1 = P_{best} + r(P_{best} - P_{worst}) \tag{4.7}$$

$$C_2 = P_{best} \tag{4.8}$$

4.2.1.3 Mutation types

The last genetic operator, which the newly generated population must pass through, is the mutation operator. The two user inputs are the probability that a mutation will occur and the desired type of mutation. The probability that a mutation will occur should be small, typically less than 0.05 (5%). When the mutation probability is set too high the genetic algorithm will start to resemble a simple random search. If the user does not wish to make use of the mutation operator, a probability of 0 can be entered.

Allowing the genetic algorithm to utilize a mutation operator has several advantages. Mutations help maintain genetic diversity in a population as it ages. Often times, after many generations the population tends to lose genetic diversity and stagnate near local minimums. The mutation operator helps to prevent this from happening by randomly mutating genes in the chromosomes. By introducing (or in some cases reintroducing) changes in a chromosome or individual gene it is possible for better more fit individuals to appear. In this algorithm the mutation operator changes the value of one randomly selected gene (i.e. variable) in the chromosome, which in some situations can greatly improve the individuals fitness.

As with the crossover operator several separate types of mutations have been implemented. In practice, certain mutation types may work best for each type of problem, so the user should experiment with different crossover and mutations types to see which combinations work best. A total of 3 mutation types have been implemented in the hybrid algorithm, which can operate on both integer and real valued genes.

Uniform Mutation The uniform mutation operator replaces the value of a randomly chosen gene with a uniform random value between the variable's upper and lower bounds. The advantage of this operator is that it allows the population's genetic diversity to be maintained by inserting a random gene that other mutation methods may not be able to obtain. Similar non-uniform mutation, based on other distributions, can be used as well.

Boundary Mutation With boundary mutations, a single, randomly selected, variable is changed to either the variables upper or lower bounds. The choice to use either the upper or lower bound is done with a virtual coin flip. This method best promotes genetic diversity when used in combination with either the arithmetic or heuristic crossover operators.

Sliding The sliding mutation operator is similar in performance to uniform mutations. A randomly selected gene is multiplied by a randomly selected value between 0.8 and 1.2. In essence, this operator allows the gene to be slightly mutated by $\pm 20\%$. The operator then checks to ensure that the variable bounds have not been violated. If a violation occurs the variable is set to the closest bound.

4.2.2 Non-Linear Programming (NLP) Solvers

Evolutionary algorithms, particularly genetic algorithms, are well suited for global optimization. However, when genetic algorithms are used to optimize impulse multiple gravity-assist and low thrust problems they often only find solutions near the global optimum. Alternatively, non-linear programming (NLP) solvers typically only converge to locally optimal solutions. By modifying the genetic algorithm to utilize an NLP solver to determines locally optimal solutions a robust global optimization algorithm can be developed. This hybrid algorithm, known as the GNLP optimization algorithm, is able determine near globally optimal solutions by combining the global convergence of the genetic algorithm with accuracy of the nonlinear programming algorithm. The proposed algorithm is able to efficiently solve complex problems by significantly reducing the population size and number of generations required to converge on near-globally optimal solutions.

The GNLP algorithm should only be used to optimize functions that are continuous and at least twice differentiable, at least in the neighborhood of the proposed solution. Because of this the NLP solver does not iterate on integer variables, which often introduce large discontinuities. The genetic algorithm is used exclusively to optimize integer variables. These properties make optimizing both high an low thrust mission design problems good candidates for the proposed GNLP optimization algorithm.

4.2.2.1 UNCMIN based non-linear programming solver

The NLP solver implemented in the hybrid algorithm is based on the UNCMIN optimization algorithm, originally written in FORTRAN 77, and introduced in [56, 57]. This algorithm is a quasi-newton algorithm based on steepest descent methods, which requires only the objective function values. However, to improve convergence and run times this algorithm does allow user supplied gradient and Hessians. The NLP algorithm implemented in the hybrid algorithm is an updated version of the original UNCMIN algorithm written in Fortran 90. Additional information on this algorithm can be found in [56] and [57].

As implemented, the UNCMIN algorithm is an extremely modular system of algorithms capable of multiple step selection strategies and multiple derivative calculation techniques (for both 1^{st} and 2^{nd} order derivates). First order derivatives can be calculated analytically, or by forward or central finite difference. The second derivative Hessian matrix can be calculated analytically, with a Broyden-Fletcher-Goldfarb-Shanno (BFGS) approximation [58–61], or by finite difference. The BFGS method approximates the Hessian from gradient information and is often used in quasi-Newton methods. The BFGS method gives a sufficiently accurate approximation, for all the problems in this dissertation, for the Hessian matrix, which is computationally expensive to evaluate numerically. Utilizing this approximation significantly increases the efficiency of the final algorithm. Having all of these options allow for a very robust algorithm that can be adjust by the user to work for a large variety of problems. For the MGA and MGA-DSM problems gradients are calculated with the forward finite difference method, while the Hessian matrices are calculated with the BFGS approximation. Calculating the Hessian via finite difference increased the solution accuracy, but also severely increased the computational requirements of the NLP solver. For the preliminary trajectory design problems, using the BFGS approximation doesn't have a noticeable effect the convergence of the hybrid algorithm when compared to the more accurate finite difference Hessians. Near locally minimum solution, the Hessian approximation gives sufficiently accurate results, while minimizing the computations the NLP solver must perform.

4.2.2.2 CONMIN based non-linear programming solver

In addition to the UNCMIN unconstrained optimization algorithm the classical CONMIN constrained optimization has also been implemented. This is necessary for problems such as low-thrust mission optimization problems, which often have match-point constraints that must be met in order to determine feasible trajectories. Like the UNCMIN algorithm, this algorithm was originally written in FORTRAN 77, but has since been updated to modern Fortran. This algorithm can determine locally optimal solutions to problems that have linear and non-linear constraints.

The basic method of optimization utilized in this algorithm is the method of feasible directions. Like the UNCMIN algorithm, only a user supplied objective function, which calculates the objective function and problem constraints is required. The algorithm can calculate derivatives analytically with user gradient functions or numerically via finite difference methods. This algorithm should only be used with problems that have a relatively low dimensionality. Further information on the CONMIN optimization algorithm can be found in [62] and [63].

4.2.2.3 COBYLA based non-linear programming solver

In addition to the CONMIN constrained optimization algorithm a second constrained optimization algorithm, known as COBYLA, has also been implemented in the algorithm [64, 65]. This NLP algorithm constructs linear polynomial approximations to the constraint and objective function using a simplex method [65]. By utilized a simplex method the derivatives can be approximated without the need to compute derivatives via finite difference. For simpler problems, or if an approximate solution neighborhood to the optimal solution is known this algorithm can give superior performance to the CONMIN solver. For highly nonlinear problems, the CONMIN NLP solver often determines local minima more efficiently.

4.2.3 Hybrid Algorithm Implementation

The final hybrid GNLP algorithm is able to achieve significant performance increases, in terms of algorithm efficiency, over the standard GA implementation. This is possible because

each individual population member is optimized with the NLP solver, rather than simply evaluating the objective function and relying on the GA for the entire optimization process. Through this process the GNLP algorithm is able to combine the global convergence properties of the genetic algorithm with the local solution accuracy of the non-linear programming solver.

With the hybrid GNLP algorithm the integer and real valued variables are arranged into separate, but linked chromosomes. This is done to simplify the drivers for the NLP algorithms and because the NLP solvers utilized by the hybrid algorithm are only able to iterate upon the real valued variables. By separating out the chromosomes the real valued chromosomes are optimized by both the selected NLP solver and the genetic algorithm while the optimization of integer variables is performed strictly by the genetic algorithm.

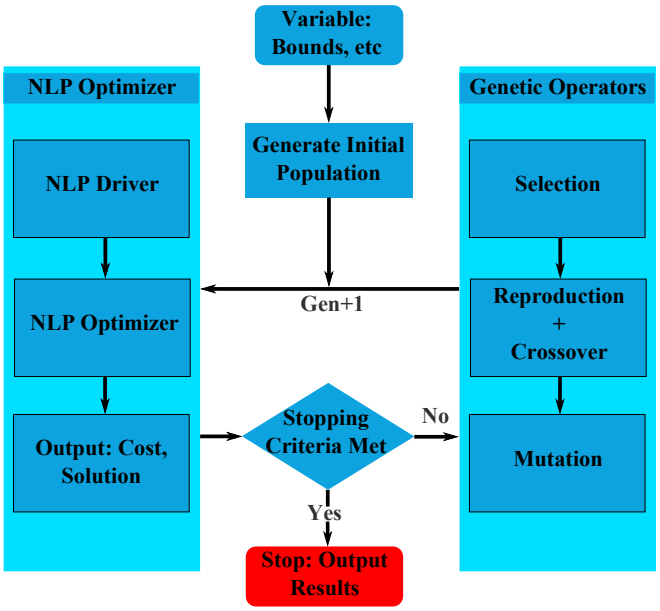


Figure 4.5 Flow chart for hybrid GNLP algorithm.

A flow chart of the hybrid GNLP algorithm is shown in Fig. 4.5. This algorithm flow is similar to the genetic algorithm, except the objective function evaluation step has been replaced by the NLP optimization step. In this step a driver for the NLP solver is called to set the NLP user inputs for the particular problem. The NLP solver then determines a locally optimal solution and outputs the results to the genetic algorithm. After this the genetic operators create a new population and the process is continued until the algorithm exits and outputs the

final results. Which NLP algorithm is used is one of the user input of the final algorithm. If the problem does not have strict constraints that are commonly violated the UNCMIN algorithm should be used.

4.3 Benchmarking the Optimization Algorithms

Before using the GNLP algorithm for MGA, MGA-DSM, and low-thrust transcription mission design optimization is it useful to test the performance of the various algorithm options on standardized benchmark global optimization functions. By testing the various optimization methods utilized by the GNLP algorithm the performance of each solution method can be tested prior to determining solutions for the MGA, MGA-DSM, or low-thrust mission optimization problems. This analysis also helps determine which solutions methods are appropriate for a given type of problem.

4.3.1 Benchmark Test Functions

A total of eight test functions have been evaluated in order to determine the performance of the various options for the optimization algorithm. This set of test functions is taken from various literature sources [42, 66, 67] and represents problems with a varying degree of difficulty, and dimensionality (2 to 15 variables).

4.3.1.1 Branin's function

The Branin function is an optimization test function that consists of two variables and is defined as

$$f = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_i - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10 \tag{4.9}$$

This function has a total of three globally optimal solutions within the bounds $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$. A surface plot of the function values is shown in Fig. 4.6.

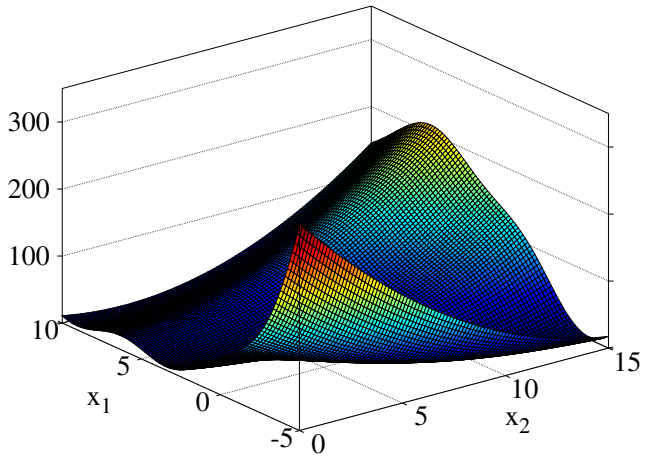


Figure 4.6 Branin's test function.

4.3.1.2 Six-Hump camel function

The second optimization test function considered is known as the six-hump camel function. As the name suggests, this function has four local minima and two global minima. The six-hump camel function is a function of two variables and is defined as follow:

$$f = \left(\frac{x_1^4}{3} - 2.4x_1^2 + 4\right) x_1^2 + \left(4x_2^2 - 4\right) x_2^2 + x_1 x_2 + 2 \tag{4.10}$$

where the two variables are bounded as: $x_1 \in [-3, 3]$ and $x_2 \in [-2, 2]$. A plot of this function is shown in Fig. 4.7.

4.3.1.3 Goldstein-Price function

The Goldstein-Price optimization test function is a function of two variables which has three local minima and one global minimum. This function is calculated as

$$f = ab \tag{4.11}$$

where a and b are calculated from the following:

$$a = (x_1 + x_2 + 1)^2 \left(3x_1^2 + 3x_2^2 + 6x_1x_2 - 14x_1 - 14x_2 + 19\right) + 1$$
$$b = (-3x_2 + 2x_1)^2 \left(12x_1^2 + 27x_2^2 - 32x_1 + 48x_2 - 36x_1x_2 + 18\right) + 30$$

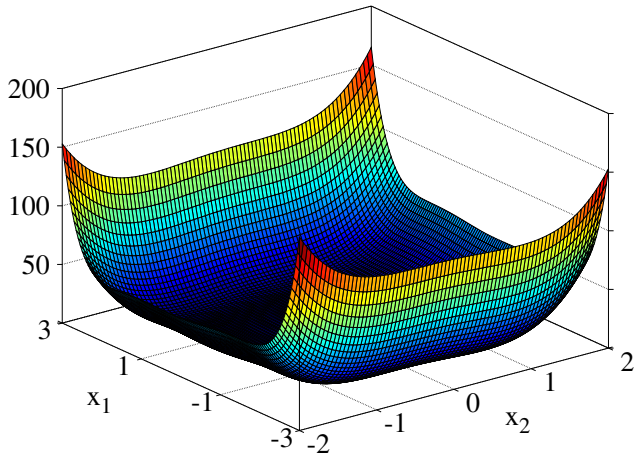


Figure 4.7 Six-hump camel test function.

The test area is typically bounded as: $x_1 \in [-2, 2]$ and $x_2 \in [-2, 2]$. A surface plot of this function is shown in Fig. 4.8.

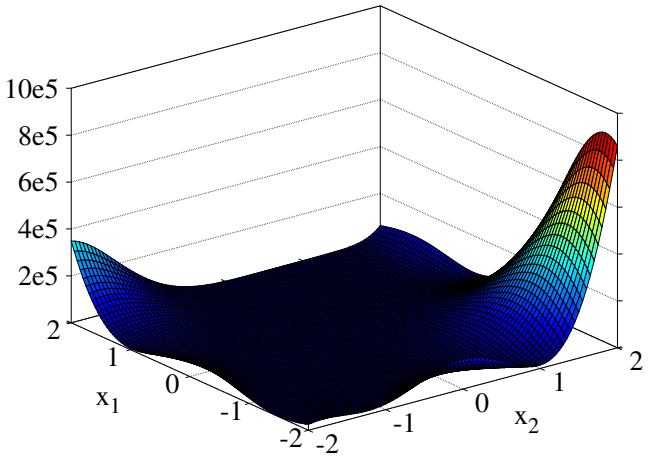


Figure 4.8 Goldstein-Price test function.

4.3.1.4 Shubert's function

Shubert's function is a two dimensional multimodal optimization test function. This function has numerous local minima and 18 global minima in the range $x_i \in [-10, 10]$. Shubert's function is calculated as

$$f = \sum_{i=1}^{5} \left(i \cos\left[(i+1)\, x_1 + i\right]\right) \sum_{i=1}^{5} \left(i \cos\left[(i+1)\, x_2 + i\right]\right) \tag{4.12}$$

A surface plot of Shubert's function is shown in Fig. 4.9. For clarity only the regions within the variable bounds of -2 and 2 is shown.

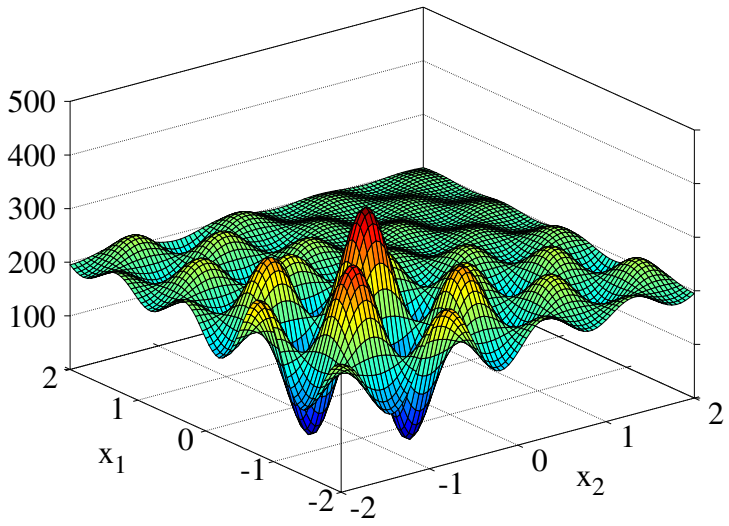


Figure 4.9 Shubert test function.

4.3.1.5 Rastrigin's function

Rastrigin's function is a multidimensional optimization test function. This function is highly multimodal, but contains only one global minimum[67]. This test function has numerous regularly spaced local minima. However, this doesn't give any advantage for the optimization methods utilized in the GNLP algorithm. Rastrigin's function is defined as

$$f = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cos\left(2\pi x_i\right)\right) \tag{4.13}$$

where n indicates the dimensionality of the problem to be optimized. For the tests in this section a dimension of $n = 10$ was used. The bounds for this problem are: $x_i \in [-5.12, 5.12]$. A two dimensional plot of Rastrigin's function is shown in Fig. 4.10.

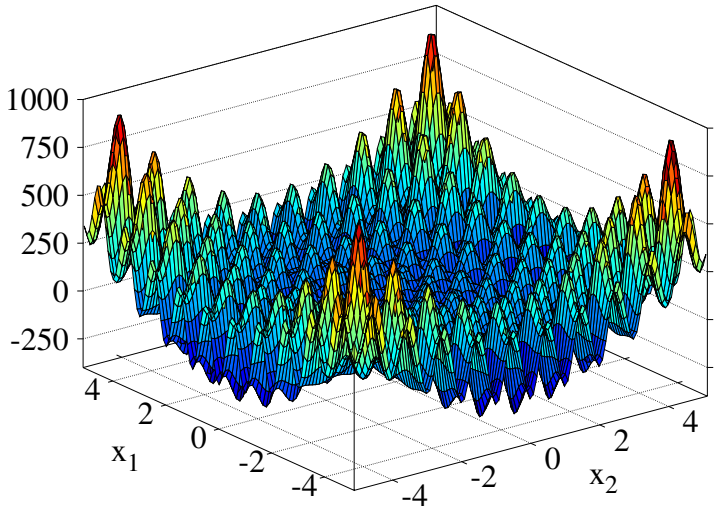


Figure 4.10 Rastrigin's test function.

4.3.1.6 Rosenbrock's function

Rosenbrock's function is a classic optimization problem. This problem is multidimensional with a single global optimum inside a long parabolic shaped valley, which has large gradients near the boundary and very small gradients near the valley floor. This problem is important because mission design problems, especially to multiple gravity-assist problems, because they often contain similarly shaped valleys [42]. Rosenbrock's function is defines as

$$\sum_{i=1}^{n-1} \left(100 \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right) \tag{4.14}$$

where n is the dimensionality of the problem. For this functions the same bounds as Rastrigin's function are used. The two dimensional surface plot of Rosenbrock's function is shown in Fig. 4.11.

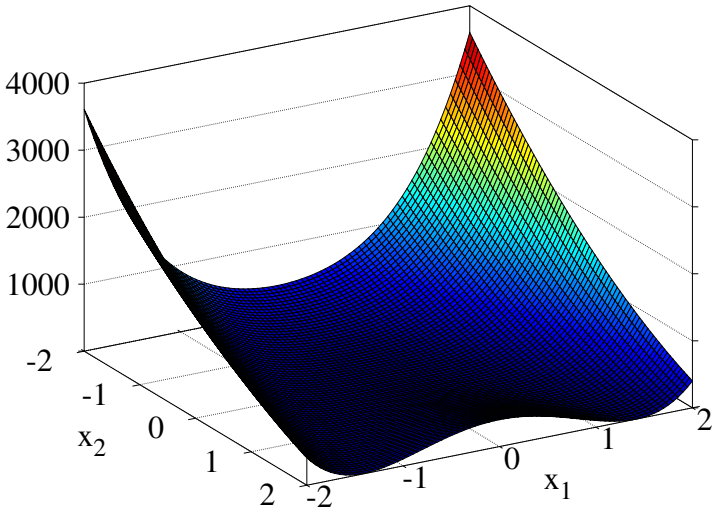


Figure 4.11 2-dimensional Rosenbrock test function.

4.3.1.7 Shekel foxhole function

The Shekel foxholes function is a multidimensional test function of up to 4 dimensions. In this study the Shekel 10 version of the function is utilized, which has 9 local minima and a single global optimal solution. The function is calculated as follows:

$$f = -\sum_{i=1}^{m} \left(\sum_{j=1}^{n} (x_i - C_{ij})^2 + \beta_i \right)^{-1} \tag{4.15}$$

where n is the dimensionality of the problem and m can be varied, but is set to 10 for this study. The $\boldsymbol{\beta}$ and $\mathbf{C}$ constants fixed in advance as

$$\boldsymbol{\beta} = \frac{1}{10}\,[1, 2, 2, 4, 4, 6, 3, 7, 5, 5]^T$$

$$\mathbf{C} = \begin{bmatrix} 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3 \\ 4 & 1 & 8 & 6 & 3 & 2 & 5 & 8 & 6 & 7 \\ 4 & 1 & 8 & 6 & 7 & 9 & 3 & 1 & 2 & 3 \end{bmatrix}^T$$

For this problem the bounds are defined as: $x_i \in [-10, 10]$, and the two dimensional Shekel 10 function is shown in Fig. 4.12.

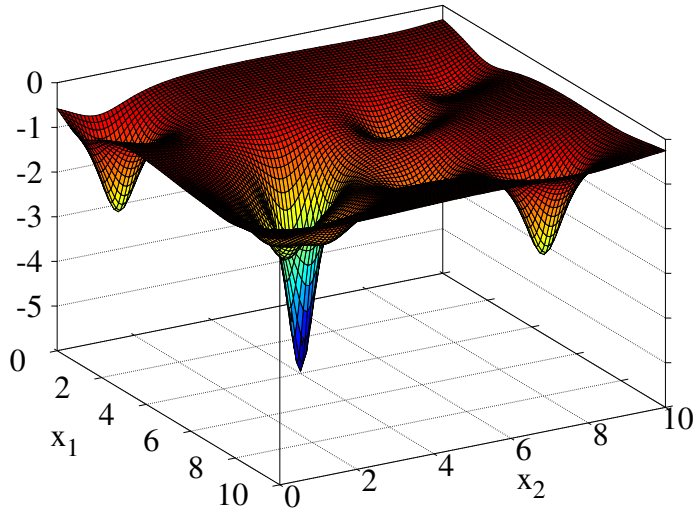


Figure 4.12 2-dimensional Shekel test function with $m = 10$ and $n = 2$.

4.3.1.8 Fletcher-Powell function

The Fletcher-Powell function is a highly multimodal function first introduced by Fletcher and Powell in 1963 [68]. This function is not symmetric and has many randomly distributed minima. This random placement of minima is achieved through the use of two random matrices, $\mathbf{A} = a_{ij}$ and $\mathbf{B} = b_{ij}$, and a random solution vector, $\boldsymbol{\alpha} = \alpha_j$. The function is determined as follows:

$$f = \sum_{i=1}^{n} (A_i - B_i)^2 \tag{4.16}$$

where A_i and B_i are calculated as

$$A_i = \sum_{i=1}^{n} (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j)$$

$$B_i = \sum_{i=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos x_j)$$

This function has 2^n minima and maxima within the the bounds $x_i \in [-\pi, \pi]$. In addition, the **A** and **B** matrices are chosen randomly within the bounds, $a_{ij}, b_{ij} \in [-100, 100]$, while the solution vector, $\boldsymbol{\alpha}$, is randomly chosen within the same bounds as x_i.

4.3.2 Test Results

The 8 functions were each tested with all 4 possible optimization methods in the GNLP algorithm, which include the standard genetic algorithm, and the UNCMIN, CONMIN, and COBYLA NLP solvers. Each algorithm was run until it was within a tolerance of $1\text{x}10^{-5}$ of the global minimum for each problem. Because the GNLP algorithm is a stochastic optimization algorithm each test was repeated 10 times in order to determine the number of function evaluations for each test problem and solution method combination. To pass the tests, each solver option had to determine the optimal solution for each of the 10 tests. This was achieved by varying the population size, while keeping every all other parameters the same.

The performance of each GNLP solution method for the benchmark problems are shown in Tables 4.2 through 4.10. Each table shows the minimum, maximum, and average number of function evaluations for each optimization method utilized by the GNLP algorithm.

The UNCMIN solver typically requires the least function evaluations, followed by the CONMIN, COBLYA, and genetic algorithm sovlers respectively. If the optimization problem can be formulated as and unconstrained problem, or if the constraints can be incorporated into the objective function the UNCMIN solver will likely provide the best results.

The pure genetic algorithm is good at finding the general solution neighborhood, however, is has difficultly providing the required solution accuracy for multidimensional problems. This is especially true for the Rosenbrock and Fletcher-Powell test functions, as shown by Tables 4.7, 4.8, and 4.10. For these complex high dimensional problems pure genetical algorithm optimization requires 4 orders of magnitude higher number of function evaluations than both the UNCMIN and CONMIN solvers. For most of these problems, the hybrid algorithm with each of the three NLP solvers was able to determine solutions within a few generations with population sizes as small as 4. The pure genetic algorithms often required population size in the

50-100 range run for a large number of generations. This is especially true for the Rosenbrock and Fletcher-Powell functions.

For medium difficultly problems with approximately linear constraints the COBYLA optimization algorithm will likely give good performance. For complex problems with highly nonlinear constraints the CONMIN solver should be used.

Table 4.2 Benchmark results for Branin's function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	1,020	28	287	102	√
Max	5,470	112	332	402	√
Avg	2,433	65	313.6	196.5	√

Table 4.3 Benchmark results for the six-hump camel function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	1,519	84	274	106	√
Max	1,877	344	490	120	√
Avg	1,678	186	397	113	√

Table 4.4 Benchmark results for the Goldstein-Price function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	1,524	1,583	6,184	5,138	√
Max	13,135	2,271	6,612	16,754	√
Avg	7,332	1,944	6,448	8,625	√

4.4 Conclusions

As the results show, for these benchmark problems, the hybrid GNLP optimization algorithm is able to converge on global optimal solutions with significantly fewer function evaluations than the pure genetic algorithm. The GNLP algorithm can employee various nonlinear programming solvers, making it capable of determining globally optimal solutions for a variety of optimization problems. In the following chapters the GNLP algorithm will be used to determine optimal trajectories for the MGA, MGA-DSM, and low-thrust transcription mission

Table 4.5 Benchmark results for the Shubert's function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	2,282	84	464	218	√
Max	26,926	2,728	2,728	1,844	√
Avg	7,933	937	1,601	752	√

Table 4.6 Benchmark results for the Rastrigin 10-dimensional function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	37,000	15,999	48,161	47,072	√
Max	76,973	85,791	162,804	114,436	√
Avg	54,833	37,777	93,589	77,261	√

analysis problems. It will be shown to provide optimal (or near optimal solutions) for some of the most complex problems in astrodynamics today.

Table 4.7 Benchmark results for the Rosenbrock 10-dimensional function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	33,678,057	2,550	6,009	12,427	√
Max	46,549,054	8,690	9,699	55,049	√
Avg	36,917,425	6,527	8,355	29,547	√

Table 4.8 Benchmark results for the Rosenbrock 15-dimensional function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	93,095,900	4,001	12,612	63,537	√
Max	93,103,581	6,151	17,918	163,527	√
Avg	93,100,550	5,248	14,839	122,380	√

Table 4.9 Benchmark results for the Shekel-10 function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	6,291	194	690	614	√
Max	22,238	1,258	2,307	2,087	√
Avg	13,331	724	1,002	871	√

Table 4.10 Benchmark results for the Fletcher-Powell 10-dimensional function.

	GA	UNCMIN	CONMIN	COBYLA	Global Min
Min	12,611,968	6,351	6,348	31,722	√
Max	18,811,275	65,454	313,106	90,302	√
Avg	16,794,209	18,731	75,706	59,285	√

CHAPTER 5. COMPUTATION OF MULTIPLE GRAVITY-ASSIST AND IMPULSIVE DELTA-V MANEUVER MISSIONS

In this chapter the hybrid GNLP algorithm is utilized to determine optimal/near optimal trajectories for both multiple gravity-assist missions (MGA) and multiple gravity-assist with deep-space maneuvers missions (MGA-DSM). It will be shown that the GNLP algorithm is able to efficiently and robustly determine optimal solutions for these types of mission in and automated fashion.

In addition this algorithm will be shown to offer advantages over other common approaches to optimizing these types of missions. While other approaches often separate the optimization of the mission specific variables (time-of-flights and other variables) and the number of gravity-assists/planetary flyby order [40, 69], the GNLP optimization algorithm can optimize both the mission design variables and number of gravity-assists/planetary flyby order, which results in efficient and robust optimization of MGA and MGA-DSM missions. Other research groups have also optimized all of the MGA and MGA-DSM mission parameters with evolutionary algorithms [36–38], but the hybrid GNLP algorithm often determines trajectories with a lower total required ΔV and a lower total computational cost.

5.1 Introduction

Since the launch of Sputnik 1, on Oct. 4th, 1957, the human race has been driven by curiosity to explore beyond Earth and enter into the solar system. NASA's Mariner 2 spacecraft, launched on Aug. 27th, 1962, was the first successful interplanetary space probe, passing within 22,000 miles of Venus. Since then a myriad of spacecraft have been launched in an effort to explore our solar system, although only a fraction of our solar system has been explored. To

further explore the solar system larger spacecraft and scientific payloads will be required. These missions can only be enabled through increasingly complex trajectories, often by the use of a combination of planetary gravity-assists, deep-space corrections maneuver and, more recently, low-thrust solar electric propulsion.

To understand the significance of gravity-assists and their importance to interplanetary missions, it is important to understand the classical theories of space travel, as developed by notable rocket pioneers such as Hohmann [70], Goddard [71], and Tsiolkovsky [72]. Prior to the invention of gravity-assists, all spacecraft relied entirely on chemical rocket propulsion. This means there was a high energy barrier, which prohibited exploration mission beyond the Earth's moon, Mars, and Venus. In essence, the large ΔV necessary for exploring the solar system beyond our nearest neighbors is simply larger than what is feasible from traditional chemical rocket propulsion.

By the 1950s initial work in advancing interplanetary trajectories was being performed by many notable orbital mechanics specialists such as Breakwell, Battin, Lawden, and many others [1–5]. By 1959 Battin developed a method for computing round trip free return trajectories using solutions to Lambert's problem, but had stopped short of calculating the trajectories because a mathematical method to represent the 3-dimensional flyby trajectory had not yet been developed. In 1961 Minovitch [6] developed a new method to represent 3-dimensional conic orbits with two orbital vectors, commonly known as $\vec{e}$ and $\vec{h}$. With this new method he was able to develop what is now known as the *patched conic method* and began to calculate trajectories using gravity-assists. Trajectories utilizing gravity-assists are able to obtain higher orbital energies than previously possible using only chemical rocket propulsion, opening up new opportunities for further exploration of our solar system.

To this day gravity-assists are commonly used for interplanetary and interstellar missions. A few notable missions that have utilized gravity-assists include Mariner 10, Pioneer 10, Pioneer 11, Voyager 1, Voyager 2, Galileo, Cassini, and Messenger. The Voyager and Pioneer missions used gravity-assists to insert the spacecraft on solar system escape trajectories, while Mariner 10, Galileo, Cassini, and Messenger missions exploited inner-planetary gravity-assists to reach Mercury, Jupiter, and Saturn respectively, without the need for heavy lift launch vehicles [73–

75]. As mission become increasingly complex determining feasible trajectories quickly becomes a daunting task.

These types of interplanetary and interstellar mission design problems present significant optimization challenges. They are extremely nonlinear, often with multiple strong basins of attraction in the neighborhood of optimal solutions, and are discontinuous when the planetary gravity-assist order is varied. The traditional method for solving complex impulsive mission design problems is to have the designer prune the decision space in order to obtain acceptable solutions. However, without robust automated methods to optimize such trajectories, globally optimal solutions may be overlooked by even the most experienced mission designer. Using the proposed hybrid optimization algorithm, an autonomous mission design program has been developed. It will be shown that, when used properly, this new algorithm can find optimal or near optimal solutions in an autonomous and computationally efficient manner.

The proposed hybrid optimization algorithm combines stochastic evolutionary optimization techniques with traditional gradient based non-linear programming methods to develop an optimization algorithm than can determine near-globally optimal interplanetary trajectories. The new algorithm is capable of finding near-optimal solutions for complex missions with multiple flybys and impulsive ΔV maneuvers, similar to NASA's Galileo and Cassini missions [76, 77]. This new algorithm is also capable of optimizing the number of gravity-assists and planetary flyby order. The results will be compared with the past Galileo and Cassini missions, in an effort to validate both the problem formulation and the hybrid optimization algorithm itself. The ultimate goal is to develop mission design tools that can efficiently and autonomously determine near optimal trajectories.

Two classes of problems are developed and tested. The first are traditional multiple powered gravity-assist(MGA) missions, such as the Voyager or Galileo missions. The second mission type uses multiple gravity-assists with the option for a deep space maneuver during each interplanetary leg of the mission. This mission type is often referred to as the MGA-DSM model and was required for the Cassini mission [69, 75].

5.2 Problem Formulation

Depending on mission requirements, trajectories can be modeled by either two-impulse or three-impulse trajectories. Missions that don't require large deep space maneuvers (DSMs) can be modeled with the two-impulse model, commonly known as the multiple gravity-assist (MGA) model. Mission such as Mariner 10, Pioneer 10/11, Voyager 1/2, and Galileo can all be modeled with the MGA model. These types of missions require far fewer variables than the three-impulse MGA-DSM model, allowing significantly decreased computational costs when compared to missions requiring three-impulse transfers.

The three-impulse, or MGA-DSM, model is an extension of the simple two-impulse model. This model simply propagates the orbit by a given amount of time before performing a deep space maneuver to target the new planet in the trajectory sequence. The purpose of outlining both of these methods is to formulate a way to determine objective functions for the optimization algorithm. The objective functions typically consists of required mission ΔV's, the Earth departure V_∞, the final arrival V_∞, and any other mission constraints. Any permutation of these elements can be used to meet specific requirements for each mission.

5.2.1 Multiple Gravity-Assist Model

With the multiple gravity-assist (MGA) two-impulse model planetary flybys are modeled using the 3-dimensional patched conic method developed by Minovitch [6]. The flyby orbit geometry is defined by the incoming and outgoing spacecraft velocity vectors, which are given by two solutions to Lambert's problem [78]. This *patching* results in a powered hyperbolic orbit for each gravity-assist that requires a ΔV at the perigee of each flyby. It should be noted that the ΔV required for each gravity-assist is typically driven to a near 0 value, resulting in a free flyby.

For each gravity-assist, the incoming and outgoing velocity vectors (in the heliocentric frame) are given directly from solutions to Lambert's problem [9, 10, 13, 14, 18]. The incoming and outgoing velocity vectors, relative to each planet, are then found by subtracting the planet's

velocity from the incoming and outgoing spacecraft velocities, given from the solutions to Lambert's problems as follows:

$$\vec{v}_{\infty-in} = \vec{V}_{s/c-in} - \vec{V}_{\oplus} \tag{5.1}$$

$$\vec{v}_{\infty-out} = \vec{V}_{s/c-out} - \vec{V}_{\oplus} \tag{5.2}$$

The perigee radius, which is required to patch the two solutions together, is a function of the incoming and outgoing orbits. The first step is to determine the semi-major axis of the incoming and outgoing hyperbolic trajectories as

$$a_{in} = -\frac{\mu_{\oplus}}{v_{\infty-in}^2} \tag{5.3}$$

$$a_{out} = -\frac{\mu_{\oplus}}{v_{\infty-out}^2} \tag{5.4}$$

where $\mu_{\oplus}$ is the target planet's gravitational parameter.

The required turning angle for the gravity-assist is a function of the incoming and outgoing v_{∞} vectors, defined as

$$\delta = \cos^{-1}\left(\frac{\vec{v}_{\infty-in} \cdot \vec{v}_{\infty-out}}{v_{\infty-in} \cdot v_{\infty-out}}\right) \tag{5.5}$$

With this model the flyby perigee radius must be equal for both legs of the hyperbolic orbit, as described by

$$r_p = a_{in}(1 - e_{in}) = a_{out}(1 - e_{out}) \tag{5.6}$$

where e_{in} and e_{out} are the incoming and outgoing orbit eccentricities. It should be noted that two eccentricities should be greater than 1 because hyperbolic incoming and outgoing orbits

are required to avoid being captured by the planet. The turning angle δ can also be represented as the sum of the transfer angles for the incoming and outgoing orbits, represented as follows:

$$\delta = \sin^{-1}\left(\frac{1}{e_{in}}\right) + \sin^{-1}\left(\frac{1}{e_{out}}\right) \tag{5.7}$$

Equations (5.6) and (5.7) can be written into a single equation that can be iteratively solved in order to determine the incoming and outgoing eccentricities, as follows:

$$f = \left(\frac{a_{out}}{a_{in}}(e_{out} - 1)\right)\sin\left(\delta - \sin^{-1}\left(\frac{1}{e_{out}}\right)\right) - 1 = 0 \tag{5.8}$$

The above equation, which is a function of the unknown parameter e_{out}, must be solved with an iterative root-finding method. For this problem, a simple Newton root-finding method works well. To start the Newton iteration an initial value for e_{out} of 1.5 works well. In a Newton iteration scheme a while loop is used until e_{out} stops changing within a specified tolerance. The number of iterations should also be monitored, so the process can be terminated if a solution doesn't converge. The required first derivative of f with respect to e_{out} is:

$$\frac{df}{de_{out}} = \left(\frac{a_{out}}{a_{in}}e_{out} - \frac{a_{out}}{a_{in}} + 1\right)\frac{\cos\left(\delta - \sin^{-1}\dfrac{1}{e_{out}}\right)}{e_{out}^2\sqrt{1 - \dfrac{1}{e_{out}^2}}} + \frac{a_{out}}{a_{in}}\sin\left(\delta - \sin^{-1}\frac{1}{e_{out}}\right) \tag{5.9}$$

When a converged e_{out} is found the perigee radius is calculated from Eq. (5.6). Finally, the ΔV required to patch the two orbits at the perigee can be determined as follows:

$$\Delta V_{GA} = \left|\sqrt{v_{\infty-in}^2 + \frac{2\mu_\oplus}{r_p}} - \sqrt{v_{\infty-out}^2 + \frac{2\mu_\oplus}{r_p}}\right| \tag{5.10}$$

The flyby perigee radius and ΔV are functions of the the spacecraft's incoming and outgoing velocities. These are found from solutions to Lambert's problem, which are a function of planetary positions and time-of-flight. The planetary positions are also a function of time, so the only decision variables with this model are the date of Earth departure and time-of-flight

for each additional leg of the trajectory. The final cost function, C, for the MGA optimization problem is represented as a function of the decision variables, $\mathbf{X}$, as follows:

$$C = f(\mathbf{X}) + g(\mathbf{X}) \tag{5.11}$$

$$\mathbf{X} = [T_0, T_1 \dots T_f]^T \tag{5.12}$$

where T_0 is the launch date and T_i are the time-of-flights for each leg of the mission. Typical penalty function, $g(\mathbf{X})$, will be discussed at the end of this section.

The final cost functions are then constructed by summing all of the required mission ΔV's, as well as any other terms the user wishes to minimize, such as the initial departure or arrival v_∞'s. Globally optimal solutions of the MGA objective function(s) can then be found with the proposed hybrid optimization algorithm.

5.2.2 Multiple Gravity-Assist Deep Space Maneuver Model

The MGA model, while simple and easy to implement, is not suitable for all missions. In many situation allowing deep space maneuver(s), where a ΔV is applied sometime during the interplanetary coast arc, results in a significant reduction of the total required spacecraft ΔV. This implies that the optimal location for mission burns may not be at the flyby perigees. This section outlines the basics of the three-impulse MGA-DSM model [52, 69, 75].

The MGA-DSM model consists of three mission phase; Earth departure, gravity-assist(s), and the final terminal phase. For the launch phase an impulsive ΔV is applied at the Earth departure radius in a direction defined by the problem variables. The departure velocity vector is defined by three variables, the v_∞ magnitude (or alternatively C_3), and the departure right ascension and declination angles, α and β, respectively. Three additional variables are needed to completely define the departure phase, these are the launch date, time of flight from launch to the first flyby, and the burn index. The burn index, ϵ, defines a point along the trajectory in which an impulsive ΔV is applied to target the next planet in the trajectory. The burn index can range anywhere between 0 and 1, although both 0 and 1 represent a simple Lambert solution

without an additional correction maneuver. Directly specifying the departure declination has the added benefit of ensuring that all of the departure trajectories can be flown by a given launch vehicle. This is important because launch vehicles often have lowered C_3 performance when the departure declination is outside the launch vehicle's intended range. The spacecraft's initial state vector at the Earth departure is fully defined by four problem variables, the Earth departure date (T_{launch}), v_∞, α, and β, as follows:

$$\vec{r}_{s/c} = \vec{r}_\oplus(T_0) \tag{5.13}$$

$$\vec{V}_{s/c} = \vec{V}_\oplus(T_{launch}) + v_\infty \left[\cos\alpha\cos\beta\vec{I} + \sin\alpha\cos\beta\vec{J} + \sin\beta\vec{K}\right] \tag{5.14}$$

From this point, the next step is essentially the same for both the departure phase and gravity-assist phases. The spacecraft's state vector is propagated forward by a time $\epsilon_1 T_1$, at which point a deep space maneuver is applied to target the next planet. This targeting is performed by applying a solution to Lambert's problem, with the time of flight given by $(1-\epsilon_1)T_l$. The ΔV required by the deep space maneuver is the magnitude of the difference between the velocity vector after the orbit is propagated and the initial velocity vector from Lambert's solution. The spacecraft's planetary arrival velocity, also given from the solution to Lambert's problem, is then used for the next phase of the mission.

Each additional gravity-assist in the trajectory requires an additional four variables. The required variables include, the flyby radius ratio r_{pi}, b-plane insertion angle γ_i, time of flight T_i, and burn index ϵ_i. The subscript i is an index specifying individual gravity-assists. The radius ratio is multiplied by the flyby planet's radius to determine the flyby perigee radius as

$$R_{pi} = r_{pi} R_\oplus \tag{5.15}$$

Because the flyby is unpowered, the incoming and outgoing v_∞ magnitudes must be equal as

$$|\vec{v}_{\infty-in}| = |\vec{v}_{\infty-out}| \tag{5.16}$$

Similar to the the MGA model, the incoming velocity vector is given by

$$\vec{v}_{\infty-in} = \vec{V}_{s/c-in} - \vec{V}_{\oplus} \tag{5.17}$$

The orientation of the outgoing velocity vector can be determined from the problem geometry. The flyby orbit's semi-major axis and eccentricity can be determined as follows:

$$a = -\frac{\mu_{\oplus}}{v_{\infty}^2} \tag{5.18}$$

$$e = 1 - \frac{R_{pi}}{a} \tag{5.19}$$

The flyby turning angle is a function of the flyby eccentricity, given by

$$\delta = 2 \arcsin \frac{1}{e} \tag{5.20}$$

The outgoing v_{∞} direction is determined from the following equation

$$\vec{v}_{\infty-out} = v_{\infty} \left[\cos\delta\hat{i} + \cos\gamma_i \sin\delta\hat{j} + \sin\gamma_i \sin\delta\hat{k} \right] \tag{5.21}$$

where the $\hat{i}$, $\hat{j}$, and $\hat{k}$ are the b-plane unit vectors, defined as follows:

$$\hat{i} = \frac{\vec{v}_{\infty-in}}{|\vec{v}_{\infty-in}|} \tag{5.22}$$

$$\hat{j} = \frac{\hat{i} \times \vec{V}_{\infty-in}}{\left|\hat{i} \times \vec{V}_{\infty-in}\right|} \tag{5.23}$$

$$\hat{k} = \hat{i} \times \hat{j} \tag{5.24}$$

The final departure velocity is found by adding the current velocity of the planet with the outgoing v_{∞} vector, and is given by

$$\vec{V}_{s/c-out} = \vec{v}_{\infty-out} + \vec{V}_{\oplus} \tag{5.25}$$

As with the Earth departure phase, the spacecraft is propagated forward by $\epsilon_i T_i$ time. The next planet is then targeted with a solution to Lambert's problem and the required deep space maneuver is then calculated.

The terminal phase of the mission can be formulated in multiple ways. The deep space maneuver which targets the final planet is part of the last flyby, so the only addition to the objective function is typically either the arrival v_∞ or a required ΔV for insertion into a specific orbit. In the case of the Cassini mission, the orbit about the final planet is defined by a specific insertion perigee radius and orbit eccentricity.

The final cost function is similar to the MGA model. However, the total number of state variables is significantly increased. As before, $f(\mathbf{X})$ is the object function (real mission ΔV's, etc) and $g(\mathbf{X})$ is a penalty function used to enforce problem constraints. In this case, n is the number of gravity-assists required by the mission. The final cost function representation and the decision variables are defined by

$$C = f(\mathbf{X}) + g(\mathbf{X}) \tag{5.26}$$

$$\mathbf{X} = [T_0, v_{\infty-0}, \alpha_0, \beta_0, \epsilon_0, T_l, T_1 \dots T_{n+1}, \epsilon_1 \dots \epsilon_n, \gamma_1 \dots \gamma_n, r_{p1} \dots r_{pn}]^T \tag{5.27}$$

The objective functions are typically formulated to minimize the total mission ΔV, or alternatively to maximize the final spacecraft arrival mass. An example of a typical objective function, without any constraint penalties, is chosen as

$$C = v_{\infty-0} + \Delta V_0 + \Delta V_1 \dots \Delta V_n + v_{\infty-f} \tag{5.28}$$

where ΔV_i is the magnitude of the deep-space maneuvers corresponding a particular planetary flyby.

5.2.3 Problem Constraints

In optimization problems, constraints are often used to help shape the final solution and ensure only feasible solutions are found. When using optimization algorithms, constraints are

used instead of hard variable limits. This ensures all possible solutions are continuous and can be optimized with the gradient based hybrid optimization algorithm.

Common constraints for mission design problems include: flyby altitude limits, mission time constraints, and penalties designed to protect against low velocity flybys. During a low-velocity flyby the spacecraft would be captured by the flyby planet, rather than gaining velocity in the heliocentric frame. These are rare and can typically only occur at the first planet in the flyby sequence. Any additional constraint which help shape the solution can be added, as long as the constraint values are approximately the same order of magnitude at the intended objective function values. It should be noted that when using the MGA-DSM model in conjunction with the hybrid algorithm all of the variables are explicitly constrained. However, low-velocity flybys may still occur.

The MGA model often results in a perigee radius that passes through the planet or close to the planet's atmosphere. In this situation a constraint function to move the solution toward more feasible trajectories can be expressed as [69]

$$g_i(\mathbf{X}) = -2 \log \frac{R_{pi}}{k R_{\oplus}} \tag{5.29}$$

Where k is a multiplier used to define how close the spacecraft is allowed to flyby a target planet. For the Cassini mission a value of 1.05 is used. However, the Galileo mission had an Earth close approach altitude of 300 km corresponding to a k value of approximately 1.047.

The next constraint penalizes low velocity flybys. This method has also been adapted from [69]. Low velocity flybys are very rare, but solutions should still be protected from these unfeasible trajectories. The orbital energy, about the flyby planet, can be calculated as

$$E = \frac{|\vec{v}_{\infty - in}|^2}{2} - \frac{\mu_{\oplus}}{R_{soi}} \tag{5.30}$$

For the flyby orbit to be hyperbolic about the planet, E must be greater than 0. However, the sphere of influence model is an approximation, so an additional 10% margin on the incoming velocity is added. A simple penalty that is scaled inversely to the flyby velocity is used. For relatively large v_{∞} values, the penalty is near 0 and is small compared to the overall cost

function value. Alternatively, for very low v_∞ values, the penalty is large enough to influence the final shape of the solution. The flyby orbital energy, adjusted for the 10% margin and final constraint are calculated using the following two equations:

$$E = \frac{|0.9\vec{v}_{\infty-in}|^2}{2} - \frac{\mu_\oplus}{R_{soi}} \tag{5.31}$$

$$g_i(\mathbf{X}) = \begin{cases} 0 & E \geq 0 \\ \dfrac{1}{|\vec{v}_{\infty-in}|} & E < 0 \end{cases} \tag{5.32}$$

Additional penalty constraints can be added to shape the solution as desired. For MGA and MGA-DSM missions, the final constraint function is represented as the sum of each individual constraint by

$$g(\mathbf{X}) = \sum g_i(\mathbf{X}) \tag{5.33}$$

5.3 Results

5.3.1 The Galileo Mission

Galileo was launched on October 18th, 1989 on a mission to explore Jupiter. The spacecraft was launched from the Space Shuttle Atlantis using a Centaur upper stage. The combination of the spacecraft's high mass, which was over 2700 kg, and the Centaur upper stage placed tight constraint on the achievable Earth escape energy, or C_3 (which is simply v_∞^2). With the launch vehicle constraints, Galileo was forced to make use of interplanetary gravity-assists to achieve sufficient energy to reach Jupiter. Three gravity-assists were utilized, the first at Venus and the second two at Earth. The mission was further complicated by the use of a 2-year resonant orbit between the two Earth gravity-assists. The mission sequence flown by the Galileo spacecraft is often referred to as VEEGA gravity-assist mission (Venus-Earth-Earth Gravity-Assist).

The Galileo mission is difficult to reproduce because of the two year resonant orbit between the two Earth flybys. Solutions to Lambert's problem are unable to determine solutions when the orbits are co-linear. Therefore, an alternate formulation to determine the flyby trajectories

is used when the time of flight between two identical planet is within ±2 days of a resonant orbit [79]. This allows the MGA algorithm to determine resonant orbits for any solutions during the optimization process. For this example mission the GNLP optimization algorithm is able to determine an optimal trajectory with a two year resonant orbit when the EVEEJ flyby sequence was chosen by the algorithm approximately 100% of the time.

For this mission only the ΔV required by the spacecraft is optimized. The Galileo mission had an upper C_3 limit of 17 km$^2/s^2$. Trajectories with a C_3 above 17 km$^2/s^2$ require an additional ΔV from the spacecraft. The objective function is also formulated to include the launch ΔV (only if the required C_3 is above 17 km$^2/s^2$), the ΔV's required for each gravity-assist, the final orbit Jupiter insertion ΔV, and any penalties associated with the flyby perigee radii and low energy flybys. The arrival insertion burn is used to insert the spacecraft into a highly elliptical trajectory around Jupiter. The same Jupiter insertion orbit as the Galileo mission is used, with an eccentricity of 0.998 and a perigee radius of 286,000 km (the apogee is 10,776,000 km) [73].

Table 5.1 Problem bounds for Earth to Jupiter MGA mission.

	N_{GA}	p_i	T_0	T_i
U_b	8	4	12/31/1992	1500
L_b	2	1	1/1/1988	25

To determine optimal trajectories for the Earth to Jupiter mission the GNLP algorithm was used to determine the number of flybys, as well as the flyby order, launch date, and times of flight for each gravity-assist leg of the trajectory. To ensure that the algorithm only optimizes an individual flyby order once, each optimal solution is saved. If the algorithm attempts the same flyby sequence again, the solution is thrown out. This simple check is performed by the objective function, so no modification of the general GNLP solver is necessary. If a solution has been duplicated the objective functions assigns a large value to the trajectory and the algorithm ignores the solution and eventually completely removes the genetic material associated with that solution. This ensures that the algorithm determines a new flyby order each time the GNLP algorithm is called. The algorithm tests thousands of flyby orders during each run, but typically converges on a particular flyby order with a few generations. In this way a significant

Table 5.2 Top flyby candidates for the Galileo Earth to Jupiter mission.

Sequence	ΔV_{sc} km/s	**Launch Date**	**TOF (years)**
EVVEEJ	0.550	22-Mar-88	4.7
EVEEJ	0.558	4-Nov-89	6.4
EVVEJ	0.599	12-May-88	6.0
EMVEMJ	0.937	7-Oct-92	6.5
EVEMJ	0.956	3-Oct-89	4.9
EMEEMJ	1.085	9-Oct-92	8.8
EMEMEEJ	1.222	10-Apr-91	8.3
EVMEMVEJ	1.283	23-Aug-91	4.5
EEMEMJ	1.300	5-Jul-91	7.8
EEMEEJ	1.324	14-Jun-91	8.1

number of flyby orders can be analyzed during each run, but only one particular flyby order is found by the end of each run.

Through extensive testing it was determined that a population size of 250 is sufficient for the GNLP algorithm to determine near-optimal trajectories for each flyby sequence. The algorithm is run until no significant change (10^{-5}) in the optimal solution has occurred for 25 generations. In a given flyby sequence a solution will typically converge within 250 generations. The GNLP algorithm is run dozens of times to determine unique optimal trajectories[1]. The total execution time for parallel entire search is approximately 2 hours on a standard Dell T3500 workstation with an Intel Xenon W3520 2.67 GHz processor.

The variable limits for the launch date, number of flybys, flyby planets, and times-of-flight are shown in Table. 5.1. For this mission up to 8 gravity-assists are allowed, ranging from Mercury to Mars. A five year launch window was searched ranging from Jan. 1st, 1988 to Dec. 31st, 1992. Each time-of-flight for the flyby trajectory legs were allowed to range from 25 to 1500 days. Many of the missions found can only be considered Galileo-like missions because a total time of flight limit isn't imposed. However, all of the top ten trajectories found have a time of flight under 9 years. Trajectories with a large number of gravity-assists have potential times of flight up to 32 years.

[1]Evolutionary algorithms rarely converge on the optimal solutions with a single run, so multiple runs are used to increase the likelihood that the real optimal solution is found.

Table 5.3 Comparison of the optimal Galileo mission with the actual Galileo mission.

	Actual Galileo	**GNLP Optimal**
Earth Launch	18-Oct-89	4-Nov-89
Venus Flyby	10-Feb-90	21-Feb-90
Earth Flyby 1	8-Dec-90	7-Dec-90
Earth Flyby 2	8-Dec-92	7-Dec-92
Jupiter Arrival	8-Mar-96	30-Jan-96

Table 5.4 Calculated time-line of the optimal mission. All ΔV's are given in km/s and the C_3 is given in km^2/s^2

Planet	**Date**	C_3 km^2/s^2	**ΔV** km/s	**Altitude** km
Earth	4-Nov-89	13.54		
Venus	21-Feb-90	-	0	18004.4
Earth	7-Dec-90	-	0	4711.2
Earth	7-Dec-92	-	0	297.8
Jupiter	30-Jan-96	-	0.558	

The top 10 mission architectures found during the search are shown in Table 5.2. In addition to finding a mission close to the Galileo Earth-Venus-Earth-Earth-Jupiter (EVEEJ) flyby sequence two other missions were found with a required spacecraft ΔV under 600 m/s. The best sequence found has 4 flybys (EVVEEJ) and a time of flight of only 4.7 years. However, the optimal launch date requires a C_3 of approximately 17 km^2/s^2, which is near the maximum the Centaur upper stage can achieve with the Galileo spacecraft. The launch window for this mission scenario occurs in March of 1988, which is approximately 1.5 years earlier than the launch date of the actual mission. This mission requires a ΔV that is approximately 8 m/s less than the actual Galileo flyby order, which is well with the margin of error for preliminary mission analysis. A third possible mission, which requires a total ΔV of approximately 600 m/s, or about 40 m/s more than the calculated Galileo trajectory was also found. This trajectory replaces the first Earth gravity-assist with Venus to form an EVVEJ flight sequence.

The calculated trajectory is compared to the actual trajectory in Table 5.3. The results of the GNLP algorithm are extremely close to the actual mission shown with two notable differences. The launch date found is approximately two and half weeks later than the actual mission, which is still within the 41 day launch window for Galileo [73]. Additionally, the final

Jupiter arrival date is a approximately five weeks early than the actual trajectory. Overall the mission found by the GNLP algorithm is sufficient for preliminary mission analysis of the Galileo mission. The final itinerary for the optimized Galileo trajectory is shown in Table 5.4.

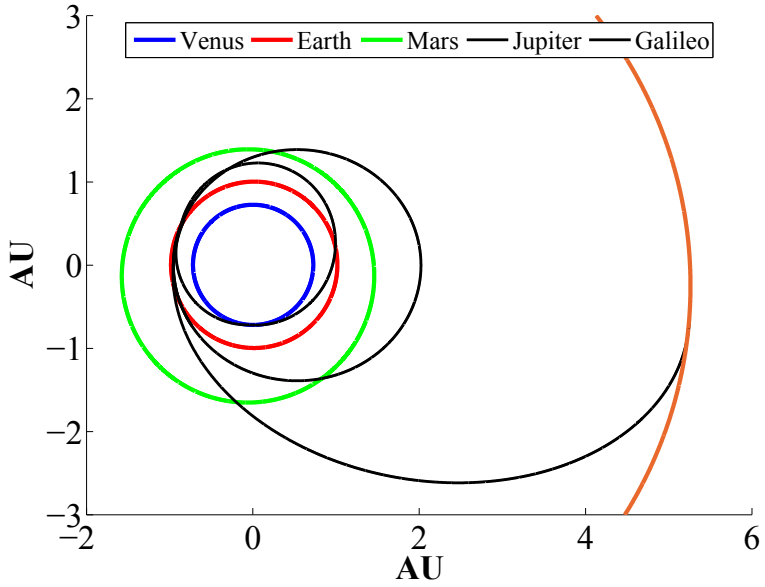


Figure 5.1 Trajectory plot for the Galileo mission.

This trajectory has a required launch where C_3 is approximately 13.5 km^2/s^2, which is the known optimal for the Galileo mission [73]. The total ΔV required by the spacecraft is approximately 560 m/s. As shown by the flight itinerary the trajectory is ballistic after launch, requiring no large maneuvers before the Jupiter orbit insertion burn. The final trajectory would likely require a few small course trajectory correction maneuvers (TCM), but this would be part of the high-fidelity mission analysis performed after the initial trajectory has been determined. At the second Earth flyby the spacecraft passes with approximately 300 km of the Earth. At first this altitude appears alarming, but it is nearly the same flyby altitude of the actual trajectory. The other two flyby altitudes are also close to the altitudes of the actual trajectory. When the spacecraft arrives at Jupiter an orbit insertion burn 558 m/s is required. This is approximately 70 m/s less than the 630 m/s burn performed by the Galileo spacecraft for the Jupiter orbit insertion. This trajectory model assumes impulsive burns, so the actual

required ΔV's will likely be larger than the calculated values because gravity losses and engine inefficiencies aren't taken into account. The final trajectory for the optimal Galileo mission is shown in Fig. 5.1.

5.3.2 Cassini Mission

The second example is to reproduce the Earth to Saturn Cassini trajectory. This trajectory requires at least one large deep-space maneuver, so the final mission must be modeled with the MGA-DSM model. Cassini was launched on October 15th, 1997 and performed four flybys (EVVEJS) prior to entering a highly elliptical Saturn orbit on July 1st, 2004. Like Galileo, the Cassini mission requires the use of gravity-assists to gain the required energy to reach the outer planets. Without at least one deep-space maneuver a feasible mission trajectory cannot be determined. Therefore, determining a trajectory to Saturn that has a required spacecraft ΔV of approximately 1 km/s requires the MGA-DSM model.

The final hybrid algorithm is able to determine optimal solutions for both the Galileo and Cassini missions. However, for the MGA-DSM problems large populations, on the order of 25,000-50,000, are required to find the optimal solutions. This requires long run times, on the order of multiple days because the NLP solver is called tens of thousands of times per generation. To improve the overall program efficiency an alternate two step formulation was used to determine optimal MGA and alternate MGA-DSM trajectories.

The idea behind the MGA-DSM model is that total required ΔV can be reduced, when compared to standard MGA missions. Therefore, initial trajectory candidates can be determined from optimal MGA missions. A search for the MGA Earth to Saturn is first performed, using the same search algorithm as the Galileo MGA search (as before only the actual required spacecraft ΔV is minimized). The results from this search are then used as the inputs to optimize the MGA-DSM missions with the GNLP algorithm. The launch dates and time-of-flights from the MGA search are used to determine the neighborhood in which the MGA-DSM search will be performed. The launch date is limited to $\pm$45 days of the optimal MGA launch date and the time-of-flights are limited to $\pm$15% of the optimal MGA mission. An outline of the basic search strategy is shown below. Using this search strategy the total execution time for the

Table 5.5 Problem bounds for MGA-DSM missions.

	C_{3L}	α	β	ϵ	γ	r_{pi}
U_b	18.1	360°	28.5°	0.95	360°	$10{,}300^2$
L_b	0	−360°	−28.5°	0.0001	−360°	1.05

entire parallel search algorithm is approximately 6 hours on the same Dell T3500 workstation as before.

1. Simple GA: Determine Gravity-Assist Order
    - Population: 250-500
    - Max. Generations: 2500
    - Variable bounds: completely open
2. GNLP: Determine final optimal solution
    - Flyby order is given from the GNLP MGA search
    - Solution neighborhood for the launch date and flight times are also given from the GNLP MGA search
    - The additional MGA-DSM variable bounds are left completely open.
    - Population: 2,500
    - Max. Generations: 2,500

The additional bounds for the MGA-DSM search are given in Table 5.5. The bounds for the initial MGA search are the same as those for the Galileo mission, except launch dates are limited from Jan. 1st, 1997 to Dec. 31st 1999. Additionally, an upper C_3 of 18.1 km^2/s^2 is used, which is the C_3 of the actual Cassini mission. If the launch dates are not constrained close to the actual Cassini mission, the algorithm converges on a solution with an EVVEJS flyby sequence requiring a ΔV of approximately 750 m/s. Unfortunately, the launch date for this trajectory occurs approximately 2 years prior to the Cassini launch.

The final GNLP algorithm and search strategy outlined above was able to reproduce the optimal Cassini trajectory. However, on approximately 50% of runs, the initial MGA search

Table 5.6 Top 20 candidates for the Earth to Saturn MGA Missions

Sequence	ΔV_{sc} km/s	**Launch Date**	**TOF (years)**
EVMVMVES	1.305	19-Jan-98	10.6
EVVEVVES	1.307	8-Feb-98	12.6
EVVES	1.411	19-Feb-98	8.4
EVMEMVES	1.524	18-Jan-98	11.6
EVVEJS	1.587	20-Oct-97	7.3
EVEMES	1.619	1-Mar-98	7.5
EMEVES	1.678	28-Aug-97	9.0
EMVES	1.717	26-Jul-99	9.1
EMEMVES	1.735	3-Aug-97	11.0
EEMVES	1.764	27-Jul-97	11.0

converges on an alternate flyby sequence. None of these alternate sequences produced MGA-DSM results better than the optimal Cassini trajectory. A list of the optimal MGA Earth to Cassini trajectories candidates is shown in Table 5.6. This list is a compilation of 5 runs of the search algorithm. The optimal mission found for an EVVEJS flyby sequence has a total required ΔV of approximately 1.6 km/s, which is approximately 600 m/s more than the actual Cassini trajectory. Four solutions were found on some of the runs that have a required spacecraft ΔV lower than the optimal EVVEJS MGA mission. When these mission were analyzed using the MGA-DSM model with the GNLP solver the best mission found was the EVVEJS flyby sequence, which is the same used in the actual Cassini mission.

A comparison of the final results with the actual Cassini trajectory is shown in Table 5.7. As this table shows, the two trajectories are nearly identical, with the optimal trajectory having a required ΔV slightly lower than the actual trajectory. Both trajectories have a large, approximately 450 m/s, deep-space maneuver during the Venus to Venus trajectory. The Saturn insertion ΔV for the actual mission is within 6 m/s of the optimal solution, which is well within acceptable margins of error for preliminary mission analysis. However, the final arrival date for the optimal trajectory is approximately 1 month later than the actual mission. This is likely because the actual mission targeted an initial tour of Saturns moons, which in not part of this analysis. The final trajectory is shown in Figs. 5.2(a) and 5.2(b).

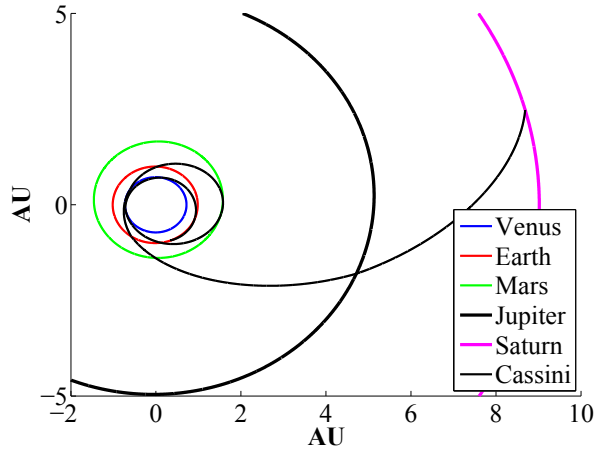
Table 5.7 Results for the Cassini mission compared to the actual results. All ΔV

Planet	**Actual Cassini**	**GNLP Optimal**
Launch Date	6-Oct-97	8-Oct-97
C_3 km^2/s^2	18.1	18.0
DSM$_1$ km/s	-	0
Venus Flyby	21-Apr-98	23-Apr-98
DSM$_2$ km/s	0.466	0.444
Venus Flyby 2	20-Jun-99	21-Jun-99
DSM$_3$ km/s	-	0
Earth	16-Aug-99	17-Aug-99
DSM$_4$ km/s	-	0
Jupiter	30-Dec-00	6-Jan-01
DSM$_5$ km/s	-	0
Saturn	1-Jul-04	2-Aug-04
Insertion Burn km/s	0.613	0.607
Total ΔV km/s	**1.079**	**1.051**

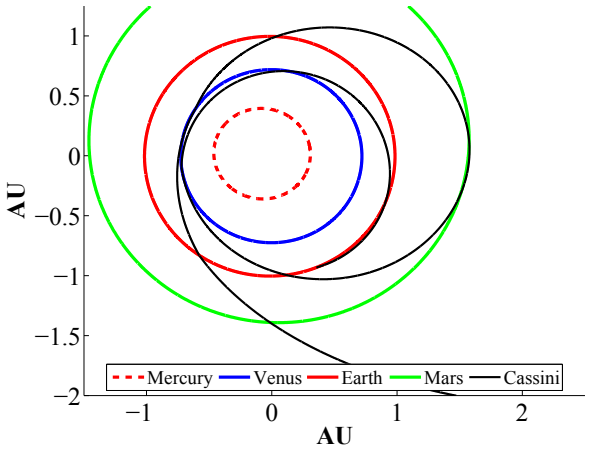
5.4 Conclusions

It has been shown that optimal preliminary interplanetary mission trajectories, with multiple possible gravity-assists, can be determined with the GNLP optimization algorithm. The GNLP algorithm combines a genetic algorithm with a nonlinear programming solver to robustly and efficiently determine near-optimal solutions to both MGA and MGA-DSM trajectories. The hybrid algorithm is able to determine the number of gravity-assists, as well as the flyby order to determine optimal trajectories for each of potential mission. With this approach thousand of possible trajectory combinations can be tested, ensuring that optimal flyby sequences are found. This GNLP based search has been applied to both the Galileo and Cassini mission to produce near optimal solutions for both test cases. For both missions the trajectory found has a lower ΔV requirement than the actual mission flown.

This new GNLP hybrid based search tool can be used for preliminary mission analysis of complex MGA and MGA-DSM missions. By utilizing this optimization algorithm, complex mission can be analyzed on standard workstations without prior knowledge of the mission structure. This includes the number of flybys and possible flyby sequences. By utilizing the



(a) Entire trajectory for the determined optimal mission.



(b) Trajectory of the inner-planetary flybys.

Figure 5.2 Trajectory for the optimal Earth to Saturn Mission.

GNLP algorithm non-intuitive missions can be found that may be overlooked during typical preliminary mission studies.

CHAPTER 6. TARGET SELECTION FOR A HYPERVELOCITY ASTEROID INTERCEPT VEHICLE (HAIV) FLIGHT VALIDATION MISSION

In this chapter the GNLP algorithm is used to optimize mission for a HAIV flight validation mission. These missions have several constraints, which are easily violated, such as approach and communication angle constraints. These constraints are used to ensure feasible trajectories are found that can be flown. Elements of the MGA and MGA-DSM missions are used to construct the trajectories, including direct intercept single spacecraft missions and dual spacecraft missions where the impact can be observed.

6.1 Introduction

Geological evidence shows that asteroids and comets have collided with the Earth in the past and will do so in the future. Such collisions have played an important role in shaping the Earth's biological and geological histories. Many researchers in the planetary defense community have examined a variety of options for mitigating the impact threat of Earth approaching or crossing asteroids and comets, known as near-Earth objects (NEOs).

As early as 1992, the idea of discovering and tracking near-Earth objects (NEOs) was proposed to the U.S. Congress [80]. That search effort, called the Spaceguard Survey, was later implemented in 1998 with the ultimate goal of finding 90% of the estimated asteroid population 1 km in diameter or larger by 2008. By focusing on only 1 km size or larger NEOs, that survey only intended to find NEOs large enough to cause global catastrophes. While not large enough to affect the entire globe, impacts by objects smaller than 1 km occur more frequently and are capable of causing significant damage. In 2005, the George E. Brown,

Jr. Near-Earth Object Survey Act expanded the original Spaceguard search to include the detection and characterization of 90% of NEOs as small as 140 m by the year 2020. To date, none of the discovered objects are predicted to be on a collision course with the Earth, but the survey still has several more years before the mission is complete. Should a new NEO be discovered on a collision course with the Earth, a mitigation effort would be necessary in order to prevent a collision with the Earth.

Given a lead time (from initial detection of the incoming NEO) of at least 10 to 20 years, depending on circumstances, various proposed technologies such as kinetic impactors, slow-pull gravity tractors, or solar sails could be employed to successfully mitigate an impact threat by deflecting the NEO's heliocentric orbit just enough to avoid a collision with Earth. When the warning time is short, nuclear technologies for a standoff, contact, or subsurface explosion may be the only viable options. However, as of the time of this writing none of the aforementioned mitigation options have been validated with a flight demonstration mission. The Asteroid Deflection Research Center (ADRC) has conducted a preliminary design for Hypervelocity Asteroid Intercept Vehicle (HAIV), a spacecraft capable of performing hypervelocity (> 5 km/s) intercept of asteroids as small as a 50-100 meters in diameter [81–83]. In this chapter a variety of mission analysis results will be presented to illustrate candidate target asteroids for a flight validation of the HAIV concept. The mission concepts considered include direct intercept missions, in which the impactor is inserted directly on an intercept course by the launch vehicle and is only allowed to perform small ΔV maneuvers prior to impact, as well as missions which allow an observer spacecraft to rendezvous with the asteroid prior to the HAIV impact.

To measure the performance and success of the HAIV it would be useful to have an observer spacecraft at the asteroid prior to the time at which the HAIV impacts the asteroid. However, due to mission and launch vehicle cost constraints it is highly desirable to perform the entire mission using one launch vehicle. Towards that end, we have designed the flight validation mission such that an observer spacecraft is not strictly required; instead, the HAIV transmits adequate telemetry to Earth for reconstruction of the asteroid impact event. The flight validation mission design is made even more cost-effective by incorporating advanced interplanetary

mission design techniques including optimally placed deep space maneuvers (DSMs) and both powered and unpowered gravity-assists via planetary flybys.

6.1.1 Previous and Future NEO Missions

To help determine the mission requirements and constraints it is useful to examine past and proposed future robotic missions to NEOs. Space agencies such ESA, JAXA, and NASA have had several successful missions that demonstrate technologies and mission capabilities that are relevant to the proposed HAIV demonstration mission, including terminal guidance targeting and/or landing capabilities. Some of the most notable missions are the Hayabusa Mission by JAXA, and the NEAR-Shoemaker and Deep Impact missions by NASA. The Hayabusa spacecraft, formerly known as MUSES-C, was sent to the asteroid 25143 Itokawa, a near-Earth asteroid (NEA) 535 $\times$ 294 $\times$ 209 m in size. While at the asteroid, the spacecraft performed two landings for the purpose of collecting surface samples, which were subsequently returned to Earth in June 2010. However, problems with the sample collection mechanism resulted in only tiny grains of asteroid material being returned. The spacecraft also had a small lander onboard, called MINERVA, which was to be guided to the surface of the asteroid. Unfortunately, the lander drifted into space and was unable to complete its mission. The NEAR-Shoemaker mission was designed to study the asteroid 433 Eros, which is one of the largest NEOs at 34.4 $\times$ 11.2 $\times$ 11.2 km in size. This spacecraft was the first to orbit an asteroid as well as the first to land on one. While the Hayabusa and NEAR-Shoemaker missions were designed to softly touch down on the surface of their respective asteroids, the Deep Impact mission was designed to do just the opposite. Approximately 24 hours prior to impact with the comet Tempel 1, which is 7.6 $\times$ 4.9 km in size, the impactor was separated from the flyby spacecraft and autonomously navigated to ensure a hypervelocity impact at a relative speed of 10.3 km/s.

More recently, ESA proposed a demonstration mission for a kinetic-impactor called the Don Quijote mission [84, 85]. The mission concept called for two separate spacecraft to be launched at the same time but follow different interplanetary trajectories. Sancho, the orbiter spacecraft, would be the first to depart Earth's orbit, and rendezvous with a target asteroid approximately

Table 6.1 Target selection criteria for the Don Quijote mission.

Orbit Characteristics	Preferred Range
Rendezvous ΔV	< 7 km/s
Orbit type	Amor
MOID	large and increasing
Orbit accuracy	well determined orbits
Physical Characteristics	**Preferred Range**
Size	< 800 m
Density	~ 1.3 g/cm^3
Absolute magnitude	20.4 - 19.6
Shape	not irregular
Taxonomic type	C-type
Rotation period	< 20 hours
Binarity	not binary

500 m in diameter. Sancho would measure the position, shape, and other relevant characteristics before and after a hypervelocity impact by Hidalgo, the impactor spacecraft. After Sancho studied the target for some months, Hidalgo would approach the target at a relative speed of approximately 10 km/s. Sancho then observes any changes in the asteroid and its heliocentric orbit after the kinetic impact to assess the effectiveness of this deflection strategy. Don Quijote was planned to launch in early 2011 and complete its mission in mid to late 2017. However, the mission concept was never realized due to higher than expected mission costs.

The selection process for the Don Quijote mission was based on a set of NEO characteristics defined by ESA's NEOMAP in Table 6.1 [86, 87]. Their analysis resulted in the selection of the asteroids 2002 AT_4 and 1989 ML. As seen in Table 6.2, 2002 AT_4 is roughly half the size of 1989 ML, but intercepting it requires a higher ΔV. A realistic deflector spacecraft would require a versatile design capable of intercepting and deflecting or disrupting either target on short notice.

One last notable future mission planned by NASA is the OSIRIS-REx asteroid sample return mission, which will return a sample from NEA 101955 (1999 RQ_{36}). This mission will launch in September of 2016 and will return the sample to Earth in September of 2023. This mission will utilize large DSMs, Earth gravity-assist (GA), rendezvous and proximity maneuvers, and an asteroid departure maneuver. The proposed HAIV demonstration mission will incorporate

Table 6.2 Properties of candidate targets considered for the Don Quijote mission.

	2002 AT4	1989 ML
Orbital period (yr)	2.549	1.463
e	0.447	0.137
i	1.5°	4.4°
ΔV (km/s)	6.58	4.46
Orbit type	Amor	Amor
MOID	large	large
Absolute magnitude	20.96	19.35
Taxonomic type	D-type	E-type
Diameter (m)	380	800
Rotational period (hr)	6	19

a combination of the knowledge gained from the development and execution of these NEO science missions.

6.1.2 Near-Earth Asteroid (NEA) Groups

For the purposes of this study, NEAs in the Atira and Amor orbit groups were considered. A comparison of NEA orbit families is shown in Fig. 6.1. NEAs in these groups all have perihelion distances < 1.3 AU, and many of them also cross the Earth's orbit at some point. The proximity of NEA orbits to Earth's orbit means that the ΔV required for intercept is usually small. As such, we expect that a number of NEAs will prove to be viable candidates for an NEA deflection/disruption flight validation mission. Apollo and Aten NEA orbits cross Earth's orbit, and in some cases this leads to lower mission ΔV requirements as compared to Atiras or Amors. On the other hand, this same fact means that any significant perturbation to the NEA's orbit could cause it to later impact the Earth. While unlikely, we do not want our demonstration of deflection technologies to cause such a thing to happen. The ESA also had this in mind when they selected the asteroids 2002 AT_4 and 1989 ML from the Amor group for the Don Quijote mission concept [84].

To preclude the possibility of inadvertently perturbing a previously harmless NEA onto an Earth collision course, we consider only Atira and Amor NEAs in our study. The Amor asteroid group is characterized by asteroids that approach the Earth, but do not actually cross

its orbit. By definition the perihelion distances of these asteroids lie between 1.017 and 1.3 AU. As of the time of this study, July 21st, 2012, there were 3398 Amor and Atira asteroids listed in NASA's NEO Program database.[1] While Amor asteroids are entirely outside of the Earth's orbit, the Atira asteroid group orbits are contained entirely within the orbit of the Earth. Because the orbits of Atiras and Amors are entirely interior or exterior to Earth's orbit, respectively, disturbances to the orbits of those asteroids are not likely to cause an impact with the Earth at any time after the mission.

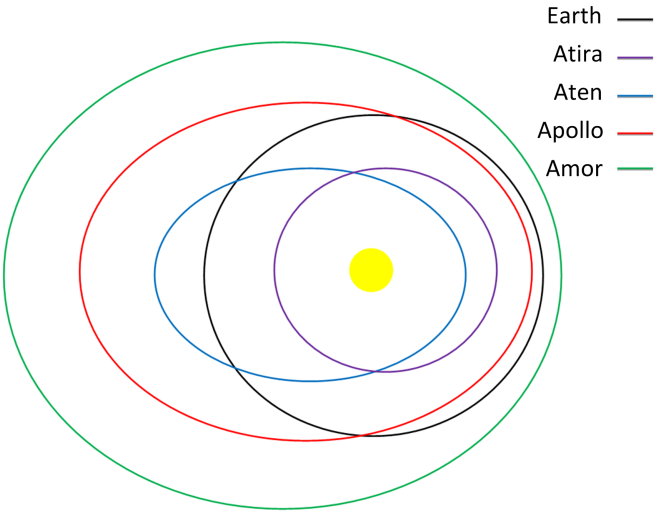


Figure 6.1 Comparison of Atira, Apollo, Aten, and Amor class asteroid orbits in relation to the Earth's orbit.

6.1.3 Mission Design Software

In addition to the HAIV concept design the ADRC has also developed mission design software tools capable of performing advanced mission analysis for thousands of potential target asteroids. Due to the large of variables in these types of missions, an exhaustive search of all 3398 asteroids would be impractical. All mission design computation will be performed using the hybrid GNLP algorithm, which utilizes both an evolutionary algorithm and traditional non-linear programming solvers. The types of missions considered in this chapter can easily

[1] http://neo.jpl.nasa.gov/stats/

be formulated as constrained optimization problems, making them ideal for hybrid genetic-nonlinear programming algorithm (GNLP). For this approach, each mission type must first be formulated as a single-valued cost function with constraints.

Utilizing a combination of evolutionary and nonlinear programming algorithms, and the modular mission design software, it is possible for multiple mission architectures for thousands of possible target asteroids to be quickly and efficiently analyzed. Given the capabilities of the mission design software several types of missions have been considered and are detailed in the following section.

6.2 Problem Formulation and Mission Constraints

The purpose of this chapter is to determine realistic mission designs for a HAIV demonstration that can be flown in the near future. Several mission constraints are imposed during the mission design survey to ensure that the resulting missions are realistic at the level of the current analysis. These constraints are enforced by the GNLP algorithm and are used to shape the solutions and ensure the best optimal solutions are found while satisfying constraints.

The first constraint imposed is designed to ensure that the HAIV spacecraft will be able to communicate with Earth ground stations during the final impact phase of the mission. This is done by adding a constraint that ensure the impact will not occur on the opposite side of the sun from the Earth. The exact angle limitations for the impact Earth-Asteroid angle are that the angle must be <175 degrees from the Earth and >185 degrees from the Earth. Fig. 6.2 illustrates the line of sight angular constraints. The second main constraint is a guidance and navigation constraint which requires that the impactor approaches from the sun facing side of the asteroid. This constraint ensures proper lighting conditions for the terminal guidance phase of the mission.

The purpose of this mission is to demonstrate the feasibility of the HAIV for the purpose of planetary defense. The HAIV spacecraft is designed for hypervelocity impacts, which are likely to be required by planetary defense missions executed with short warning time. Therefore, missions designed in this paper must have a minimum impact velocity of 5 km/s. Due to anticipated technological limitations, impact velocities over 30 km/s are penalized as well.

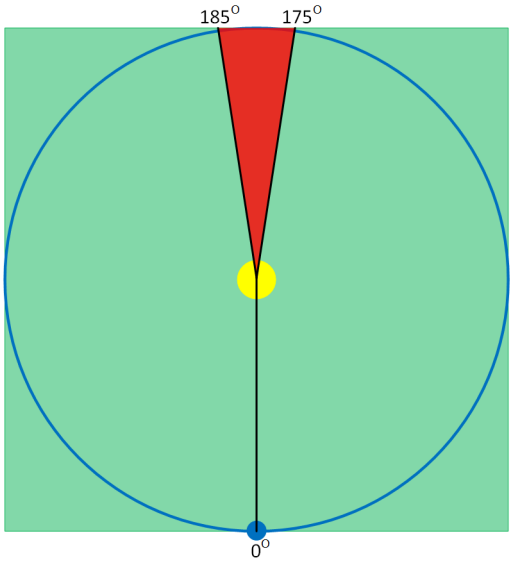


Figure 6.2 Illustration of the Earth-Sun-Asteroid line-of-site communication angle for the HAIV mission. Green indicates communicable from Earth ground stations, while red indicates area where communications are not possible.

However, it should be noted that none of the mission analyzed have had an impact velocity above 30 km/s, meaning the HAIV impactor never has to decrease its approach velocity. All other major mission constraints can be found in Table 6.3. The exact penalty functions are also discussed at the end of this section. It is worth noting that limitations on NEA orbit group and absolute magnitude (H) reduce the number of asteroids that much be searched from approximately 3500 to 1500.

Depending on mission complexity, there are different ways to formulate multiple gravity-assist problems (simple direct intercept missions are a subset of the MGA model). Simple missions, such as NASA's Voyager, Pioneer, and Galileo missions, which don't require large deep space maneuvers (DSMs), can be formulated using the MGA model. This model requires fewer variables than the three impulse MGA-DSM model, meaning they have a much lower computational cost. On many occasions, small DSMs, when compared to the departure and arrival maneuvers, can significantly lower the total overall mission cost in terms the total ΔV or maximizing arrival mass. The formulation for both types of missions are given in the previous chapter.

Table 6.3 List of mission constraints.

Asteroid Types	Amor, Atira
Earlier Launch Date	1-Jan-2018
Latest Launch Date	31-Dec-2022
Minimum Impact Velocity	5 km/s
Maximum Impact Velocity	30 km/s
H Magnitude Range	20.75-23.62
Communication LOS Constraints	>185 and <175 deg
Impact Angle Constraint	Penalized dark-side approaches
Impactor Limitations	For dual missions the impact must occur after the rendezvous S/C arrival
Maximum Departure C_3	12.5 km^2/s^2 for single S/C mission 30 km^2/s^2 for dual S/C missions

The intent of both methods is to determine a final cost function that the hybrid GNLP algorithm can optimize. That is, minimize or maximize, depending on how the individual problem is formulated. Cost functions typically consist of all of the required mission ΔV's, the Earth departure v_∞, the final arrival v_∞, and any other mission constraints. Any permutation of these elements can be used for each specific mission type analyzed in this paper. For instance in this study the departure C_3 only affects the final cost function if it is above 12.5 km^2/s^2 for the single spacecraft missions and 30 km^2/s^2 for the dual spacecraft missions. In space mission design this is often done because the launch vehicle upper stage can be leveraged for the Earth departure maneuver (typically, limited by C_3). It is worth noting that for the GNLP algorithm, used to determine the optimal mission trajectories, all mission constraints can be modeled as nonlinear constraints or applied directly to the cost function.

6.2.1 Problem Constraints

In optimization problems, constraints are often used to help shape the final solution and ensure only feasible solutions are found. When using optimization algorithms, constraints are used instead of hard variable limits. This ensures all possible solutions are continuous and can be optimized with the gradient based hybrid optimization algorithm.

Common constraints for mission design problems include: flyby altitude limits, mission time constraints, and penalties designed to protect against low velocity flybys. During a low-velocity

flyby the spacecraft would be captured by the flyby planet, rather than gaining velocity in the heliocentric frame. These are rare and can typically only occur at the first planet in the flyby sequence. Any additional constraint which help shape the solution can be added, as long as the constraint values are approximately the same order of magnitude at the intended objective function values. It should be noted that when using the MGA-DSM model in conjunction with the hybrid algorithm all of the variables are explicitly constrained. However, low-velocity flybys may still occur.

The MGA model often results in a perigee radius that passes through the planet or close to the planet's atmosphere. In this situation a constraint function to move the solution toward more feasible trajectories can be expressed as [69]

$$g_i(\mathbf{X}) = -2 \log \frac{R_{pi}}{kR_{\oplus}} \tag{6.1}$$

Where k is a multiplier used to define how close the spacecraft is allowed to flyby a target planet. For the Cassini mission a value of 1.05 is used. However, the Galileo mission had an Earth close approach altitude of 300 km corresponding to a k value of approximately 1.047.

The next constraint penalizes low velocity flybys. This method has also been adapted from [69]. Low velocity flybys are very rare, but solutions should still be protected from these unfeasible trajectories. The orbital energy, about the flyby planet, can be calculated as

$$E = \frac{|\vec{v}_{\infty - in}|^2}{2} - \frac{\mu_{\oplus}}{R_{soi}} \tag{6.2}$$

For the flyby orbit to be hyperbolic about the planet, E must be greater than 0. However, the sphere of influence model is an approximation, so an additional 10% margin on the incoming velocity is added. A simple penalty that is scaled inversely to the flyby velocity is used. For relatively large v_{∞} values, the penalty is near 0 and is small compared to the overall cost function value. Alternatively, for very low v_{∞} values, the penalty is large enough to influence the final shape of the solution. The flyby orbital energy, adjusted for the 10% margin and final constraint are calculated using the following two equations:

$$E = \frac{|0.9\vec{v}_{\infty-in}|^2}{2} - \frac{\mu_\oplus}{R_{soi}} \tag{6.3}$$

$$g_i(\mathbf{X}) = \begin{cases} 0 & E \geq 0 \\ \dfrac{1}{|\vec{v}_{\infty-in}|} & E < 0 \end{cases} \tag{6.4}$$

For the MGA model, trajectories can be found with Earth departure C_3 values that exceed the specified limit. The penalty function is formulated to represent that additional Earth departure ΔV beyond that provided by the launch vehicle's upper stage would be required from the HAIV spacecraft in order to attain the required C_3. The exact penalty function is

$$g_i(\mathbf{X}) = \begin{cases} v_{\infty-l} - \sqrt{C_{3max}} & v_{\infty-l} > \sqrt{C_{3max}} \\ 0 & v_{\infty-l} \leq \sqrt{C_{3max}} \end{cases} \tag{6.5}$$

A time constraint is used to ensure that the rendezvous spacecraft arrives at the asteroid prior to the arrival of the HAIV spacecraft and is represented as follows:

$$g_i(\mathbf{X}) = \begin{cases} 0.1(T_{rend.-arrival} - T_{impactor-arrival}) & T_{rend.-arrival} > T_{impactor-arrival} \\ 0 & T_{rend.-arrival} \leq T_{impactor-arrival} \end{cases} \tag{6.6}$$

The next constraint is added to ensure the communication line-of-sight angle is feasible during the final terminal impact phase. This is especially important in the absence of an observer spacecraft, as communications with the Earth just prior to impact will be the only way to determine mission success. The line-of-sight angle is then found as

$$LOS = \cos^{-1}\left(\frac{\vec{R}_\oplus \cdot \vec{R}_{ast}}{R_\oplus R_{ast}}\right) \tag{6.7}$$

where $\vec{R}_\oplus$ is the Earth radius vector at impact, while $\vec{R}_{ast}$ is the asteroid radius vector at impact. To ensure the LOS angle is in the right quadrant, the z component of the cross product of the two radius vectors is utilized as follows:

$$\vec{c} = \vec{R}_\oplus \times \vec{R}_{ast} \tag{6.8}$$

$$LOS = 2\pi - LOS \quad c(3) \leq 0 \tag{6.9}$$

The final penalty function for the LOS angle constraints is given in the following constraint equation:

$$g_i(\mathbf{X}) = \begin{cases} \exp\left(\frac{-1}{1-(LOS-180)^2}\right) & 175^\circ \leq LOS \leq 185^\circ \\ 0 & for\ all\ other\ angles \end{cases} \tag{6.10}$$

The final penalty function used to ensure mission feasibility penalizes the relative asteroid velocity with respect to the asteroid's position relative to the sun. This ensures that the spacecraft arrives on the sunward side of the asteroid, which results in lighting conditions that are favorable for the terminal guidance system. The asteroid radius and impactor relative velocity unit vectors are utilized to preclude numerical scaling issues. Unlike the time and LOS angle penalties, all angles greater than 0 (which corresponds to approaching directly along the asteroid-sun line from the sunward side) are penalized with a linear function. This is done because the approach angle is one of the most critical parameters to ensure mission success, as follows:

$$SA = \cos^{-1}(\vec{e}_r \cdot \vec{e}_v) \tag{6.11}$$

$$g_i(\mathbf{X}) = \frac{1}{\pi} SA \tag{6.12}$$

Additional penalty constraints can be added to shape the solution as desired. For MGA and MGA-DSM missions, the final constraint function is represented as the sum of each individual constraint by

$$g(\mathbf{X}) = \sum g_i(\mathbf{X}) \tag{6.13}$$

With the cost function for each method finalized then next step is to develop the genetic algorithm capable of optimizing these types of advanced mission design problems.

6.3 Mission Analysis Results

In this section the mission design results for the various mission architectures considered in the study are presented. Each of these mission types can be constructed using the methods described in the previous section. The hybrid GNLP optimization algorithm was then used

to optimize each constructed cost function along with all mission constraints. More details on common limits used for each variable can be found in [69, 76, 88]. For each mission type approximately 1500 potential target asteroids were scanned for feasible missions. The results presented in the following three subsections are subsets of both the standard MGA and MGA-DSM models.

6.3.1 Direct Intercept Missions

For the direct intercept mission the launch vehicle is used to inject the HAIV into an interplanetary orbit that directly impacts with the target asteroid. This is the simplest mission type analyzed, with only two variables to optimize (launch date and time-of-flight to the asteroid), which yields the largest number of feasible target asteroids. For direct intercept missions a relatively low C_3 limit of 12.5 km^2/s^2 is used. By limiting missions to low C_3 it may be possible to use a less capable (and therefore less expensive) launch vehicle than for missions that include both an impactor and rendezvous (observer) spacecraft.

With the low C_3 limit, hundreds of potential target asteroids were found during the search. These asteroid require no ΔV from the HAIV impactor, other than small statistical course corrections during the terminal impact phase of the mission. The results presented in Table 4 6.4 represent the top 3 asteroids determined by a joint study with the Mission Design Lab (MDL) at the Goddard Space Flight Center and the ADRC. These asteroids were chosen because their orbits are fairly well known (or future observations of them will be possible prior to the mission launch dates), the estimated diameters (assuming and albedo of 0.25) are close to the desired diameter of 100 meters, they have low sun approach angles, and the HAIV impactor arrival velocities are high enough to be technically challenging, yet feasible. How well the asteroid orbits are known is expressed by the Orbit Condition Code (OCC) for the orbit, which is an integer scale ranging from 0 (a very well-known orbit) to 9 (a very poorly known orbit). The OCC value is the same as the Minor Planet Center's (MPC)"U" parameter.

Table 6.4 Top 3 asteroids for the single direct intercept mission.

Asteroid	2006 CL_9	2009 QO_5	2004 BW_{18}
a (AU)	1.35	1.59	1.37
e	0.24	0.24	0.25
Diameter (m)	105	106	97
Departure C_3 (km^2/s^2)	12.0	12.5	12.5
Require S/C ΔV (km/s)	0.00	0.00	0.00
LOS Angle	349.01°	349.33°	333.17°
Sun Approach Angle	3.04°	28.05°	34.21°
Departure Date	2-Aug-19	27-Mar-19	7-Apr-19
TOF (days)	121	124	268
OCC	5	1	5
Arrival Velocity (km/s)	11.5	9.2	6.6

6.3.2 Combined Rendezvous and Direct Intercept Missions

Direct intercept missions are the baseline reference mission for the HAIV studies at the ADRC. However, it may be useful to have a spacecraft at the asteroid prior to impact. This rendezvous spacecraft would likely be a small spacecraft attached to the HAIV spacecraft that would separate at some point during the mission. The goal of the following sections is to determine a trajectory solution such that the HAIV spacecraft has a relative intercept velocity greater than 5 km/s and the observer spacecraft can rendezvous with the asteroid prior to impact. Several mission types are presented that attempt to minimize the total ΔV required by both the HAIV impactor and the rendezvous spacecraft such that the mission design can be flown using existing launch vehicles.

During initial mission analysis we determined that no feasible trajectories could be found with the relatively low C_3 used for the direct intercept mission. We therefore revised the maximum allowable C_3 to 30 km^2/s^2 and considered Delta IV and Atlas V launch vehicles.

The first mission type that we analyzed employs a single DSM after the separation of the two spacecraft. For this type of mission, referred to herein as a Type 1 mission, the rendezvous spacecraft continues on a direct intercept course, while the HAIV impactor uses a DSM to target asteroid intercept at a later date. The results for Type 1 missions are shown in Table 6.5. The lowest total ΔV required is 3.299 km/s for asteroid 2010 KU_7. However, this ΔV is

likely too high to be feasible. By examining the results it appears that the impact velocity was often driven to the lower limit of 5 km/s, which indicates that lowering the minimum required impact velocity may lower the total mission ΔV.

Table 6.5 Top 5 asteroids by total required ΔV for Type 1 missions.

Asteroid	2010 KU_7	2012 JX_{11}	2001 CK_{42}	2009 CR_4	2000 RD_{53}
a (AU)	1.67	1.97	1.42	1.75	1.79
e	0.38	0.48	0.28	0.42	0.43
Diameter (m)	95	58	266	123	280
Launch Date	14-Jul-21	7-Jun-18	18-May-18	28-Apr-21	19-Oct-22
TOF_{HAIV} (days)	668	801	543	677	652
TOF_{Rend} (days)	382	472	330	406	365.241
C_3 (km^2/s^2)	25.6	33.9	16.8	33.8	24.1
$\Delta V_{HAIV-DSM}$ (km/s)	2.387	1.990	3.258	2.153	2.030
$\Delta V_{Rend-Arr}$ (km/s)	0.912	1.186	0.453	1.247	1.725
LOS	26.2°	257.8°	128.8°	19.9°	39.9°
Sun Angle	97.8°	102.3°	93.0°	94.4°	106.0°
Impactor Velocity (km/s)	5.0	5.0	5.0	5.0	5.0
Total ΔV (km/s)	**3.299**	**3.518**	**3.711**	**3.737**	**3.755**

The second type of mission analyzed, referred to herein as Type 2, and is similar to the Type 1 mission except that the rendezvous spacecraft performs the DSM to target the asteroid at a different date than the HAIV interceptor. The top 5 results for this search are shown in Table 6.6. Examination of the results reveals that, on average, the total required ΔV is reduced by approximately 1 km/s, with the lowest required ΔV being approximately 2.6 km/s. Like the Type 1 mission results the impact arrival velocity is often driven to 5 km/s. As mentioned previously, reducing the minimum allowable impact velocity may reduce the total mission ΔV required.

6.3.3 Gravity-Assist Missions Using the MGA Model

Planetary gravity-assists are often used to reduce the required ΔV for outer planet missions, and this is also possible for the missions considered herein. Gravity-assist(s) can lower the average required total mission ΔV to the 1-1.5 km/s range, with some solutions found that require as little as approximately 600 m/s. For the following Type 3 and Type 4 missions the optimizer was used to decide which planet(s) should be used for the gravity-assist(s). In all of

Table 6.6 Top 5 asteroids by total required ΔV for type 2 missions.

Asteroid	2000 WO_{148}	2009 TV_4	1998 UM_1	1996 FO_3	2008 XB
a (AU)	1.64	1.69	1.7	1.44	1.51
e	0.38	0.37	0.40	0.29	0.31
Diameter (m)	199	59	60	212	81
Launch Date	19-Jan-20	27-Sep-20	20-Sep-18	11-Feb-22	2-Dec-21
TOF_{Rend} (days)	515	336	405	354	396
TOF_{HAIV} (days)	373	551	489	363	405
C_3 (km^2/s^2)	26.6	29.2	33.1	16.8	24.2
$\Delta V_{DSM-Rend}$ (km/s)	1.265	0.562	1.448	0.951	1.632
$\Delta V_{Rend-Arr}$ (km/s)	1.338	2.095	0.976	1.975	1.310
LOS	64.0°	198.2°	149.6°	202.8°	163.2°
Sun Angle	91.7°	101.0°	60.9°	90.1°	100.7°
Impact Velocity (km/s)	5.0	5.0	5.0	5.0	5.0
Total ΔV (km/s)	**2.603**	**2.657**	**2.709**	**2.926**	**2.942**

the best cases presented, the Earth was found to be the best planet for the gravity-assist(s). Given that most of the target asteroids have a semi-major axis between 1 and 1.3 AU, this is an intuitively satisfying result.

Type 3 missions use a single gravity-assist achieved by having the HAIV impactor perform a DSM to target the planetary flyby. The gravity-assist is then used to increase the velocity of the impactor, thus lowering the required ΔV when compared to Type 1 and Type 2 missions. Missions where the rendezvous spacecraft performs a gravity-assist to lower the arrival ΔV were also considered, however this did not improve the total required ΔV compared to Type 1 and Type 2 missions.

The last mission type, Type 4, considered for the MGA model employs two gravity-assists. The HAIV impactor targets the first gravity-assist planet with a DSM. After the first gravity-assist the spacecraft flies on a ballistic trajectory until the second gravity-assist. This resulted in total mission ΔV being reduced from the Type 3 solutions. As was the case with Type 3 missions, the optimal gravity-assist planet is the Earth for both assists. Type 4 mission results are shown in Table 6.7. For these missions the lowest required total ΔV is reduced to approximately 600 m/s, ranging up to 1.3 km/s for the top 5 asteroids.

A summary of the best Type 3 mission results is shown in Table 6.7. The best mission requires a total ΔV of approximately 1.15 km/s, which represents a nearly 1.5 km/s reduction from the best solution found for the previous mission types.

Table 6.7 Top 5 asteroids by total required ΔV for Type 3 missions.

Asteroid	2012 OO	2008 XB	2000 WO_{148}	2009 CO_5	1996 FO_3
a (AU)	1.7	1.51	1.64	1.66	1.44
e	0.38	0.31	0.38	0.35	0.29
Diameter (m)	214	81	199	120	212
Launch Date	3-Sep-21	13-Dec-21	15-Jan-20	14-Mar-22	21-Feb-22
TOF_{Rend} (days)	628	232	570	554	245
$TOF_{HAIV-Leg1}$ (days)	626	685	626	630	699
$TOF_{HAIVLeg2}$ (days)	71	227	157	717	276
C_3 (km^2/s^2)	34.5	29.7	27.6	30.4	30.5
ΔV_{HAIV} (km/s)	0.546	0.764	0.264	0.496	1.094
ΔV_{Rend} (km/s)	0.599	0.558	1.146	0.934	0.445
Gravity-Assist Planet	Earth	Earth	Earth	Earth	Earth
R_p (km)	7,398	17,249	8,033	30,908	13,098
LOS	348.5°	289.5°	316.5°	332.9°	281.7°
Sun Angle	134.7°	6.2°	157.0°	164.7 °	4.9°
Impact Velocity (km/s)	5.8	5.1	6.1	5.0	5.6
Total ΔV (km/s)	**1.145**	**1.322**	**1.410**	**1.430**	**1.539**

6.4 Conclusion

Throughout this paper several possible mission types for a HAIV demonstration mission have been analyzed. A simple direct intercept mission was found to be the best option for the HAIV flight demo mission. The direct intercept mission has the largest number of feasible candidate target asteroids, requires a minimal total post-launch ΔV (close to 0 km/s for the optimal cases presented herein), and is a representative of the sort of worst-case asteroid mitigation mission scenario that we should be prepared for. Table 6.4 shows a summary of the top 3 asteroids chosen from a search of possible target asteroids. More advanced missions, which enable an observer spacecraft to arrive at the asteroid prior to the main spacecraft impact, have also been analyzed. The results generally show that allowing the impactor spacecraft to perform multiple gravity-assists lowers the total required ΔV for each mission considered as well

Table 6.8 Top 5 asteroids by total required ΔV for type 4 missions.

Asteroid	2012 CR	2011 FR_{17}	2010 XB_{73}	2010 GZ_{33}	2011 AL_{24}
a (AU)	1.77	1.7	1.71	1.91	1.72
e	0.38	0.30	0.31	0.42	0.35
Diameter (m)	120	56	102	91	91
Launch Date	25-Feb-19	10-Mar-22	30-Nov-19	3-Apr-18	10-Jan-20
TOF_{Rend} (days)	367	406	351	325	391
$TOF_{HAIV-Leg1}$ (days)	692	703	694	683	692
$TOF_{HAIV-Leg2}$ (days)	45	569	352	1224	1126
$TOF_{HAIV-Leg3}$ (days)	605	533	540	768	565
C_3 (km^2/s^2)	26.8	29.5	30.6	30.0	32.1
ΔV_{HAIV} (km/s)	0.461	0.303	0.230	1.150	0.373
ΔV_{Rend} (km/s)	0.147	0.702	0.988	0.104	0.933
Gravity-Assist Planet	Earth	Earth	Earth	Earth	Earth
R_p (km)	34,570	32,342	40,987	7,151	19,7861
Gravity-Assist Planet	Earth	Earth	Earth	Earth	Earth
R_p (km)	28,357	87,772	21,174	20,646	91,587
LOS Angle	341.4°	54.1°	30.2°	201.4°	26.9°
Sun Angle	175.9°	168.5°	167.6°	174.9°	167.5°
Impactor Velocity (km/s)	12.3	9.5	7.2	19.4	10.2
Total ΔV (km/s)	**0.608**	**1.005**	**1.218**	**1.254**	**1.306**

as increases the impactor arrival velocity. Several feasible gravity-assist rendezvous missions are presented in Table 6.7 and Table 6.8. There are multiple possible target asteroids which require a total ΔV of 1.5 km/s or less, with the lowest combined rendezvous/impact mission ΔV of approximately 600 m/s.

CHAPTER 7. LOW-THRUST TRAJECTORY OPTIMIZATION FOR ASTEROID EXPLORATION, REDIRECT, AND DEFLECTION MISSIONS

In this chapter the hybrid GNLP algorithm is utilized to determine optimal trajectories for Asteroid Redirect Missions (ARM). It will be shown that the GNLP algorithm is able to efficiently and robustly determine optimal solutions for these mission in and automated fashion. A formulation for a low-thrust transcription method is developed to be used with the GNLP to determine low-thrust trajectories. A key modification to the Sims-Flanagan transcription [89], which is commonly used for preliminary low-thrust mission analysis and design, is introduced that utilizes a solutions to Lambert's problem to remove the midpoint match constraints. The radius and velocity match-points equality constraints are replaced with two velocity inequality constraints, which reduces the burden on the NLP solver and makes it easier to determine possible ARM candidate trajectories.

7.1 Introduction

In this chapter, we investigate the feasibility of robotically capturing and returning a near-Earth asteroid (NEA) to Earth's vicinity from a low-thrust mission design perspective. Mission design parameters will be built off of results from a feasibility study performed by the Keck Institute for Space Studies [90]. The purposes of an Asteroid Redirect Mission (ARM) are two fold. The asteroid could also be used for the purposes of astronaut activities in the vicinity of an NEA, which will be valuable for future long-term deep-space NEA missions and Mars mission. In addition to being useful as a testing ground for astronauts, natural resources found on the retrieved asteroids could be retrieved and utilized.

7.1.1 Reference Mission Design Parameters

For the study of an ARM design, several mission design drivers must be taken into account, including the required mission ΔV, total time of flight, and the mass of the returned asteroid. In addition, several assumptions are made for this study. The first is that an initial mass of 15,000 kg with a escape C_3 value of 2.0 km^2/s^2 can be obtained with out outward low thrust spiral and lunar gravity-assist. The low thrust outward spiral is assumed to be the same as the outward spiral in the original Keck study [90]. The next assumption is that a lunar gravity-assist can be utilized to capture the asteroid if the spacecraft and asteroid arrive with a maximum C_3 value of 2.0 km^2/s^2 or less [90].

The ARM missions are assumed to use two Busek BHT-20K Hall effect thrusters [91], requiring up to 40 kW of power. Each thruster has the maximum power allowance of 20 kW with the maximum thrust of 1.08 N per thruster. All the necessary parameters for this reference solar electric propulsion (SEP) system are shown in Table 7.1.

The spacecraft is initially launched into a low-Earth orbit with an Atlas V 551 launch vehicle. The low thrust Earth-departure transfer takes approximately 2.2 years to achieve a C_3 value of 2.0 km^2/s^2. The new hybrid algorithm, outlined in later sections, is then utilized to determine near optimal trajectories for both the heliocentric transfer to the asteroid and the asteroid towing transfer to the Earth's vicinity, with an arrival C_3 of no more than 2.0 km^2/s^2. The final objective function, utilized by the new optimization algorithm, is then formulated to maximize the arrival asteroid mass at the Earth arrival.

Table 7.1 A 40-kW SEP System with two Busek BHT-20K Thrusters.

Power Per Thruster	20 kW
Mass Flow Rate	40 mg/s
Maximum Thrust (each)	1.08 N
Specific Impulse	2,750 s
Thruster Efficiency	70%
Number of Thrusters	2

To narrow down the list of possible target asteroids, only asteroids with an Earth close encounter of less than 0.3 AU are considered as possible candidate. In addition to close en-

counter flyby requirement, we consider only asteroids with a maximum allowable Earth relative velocity of 3.0 km/s. With this list, two separate mission types were considered. With the first mission type, an entire small asteroid (approximately 7 m diameter, 500 tons) is returned to the Earth. The second mission type retrieves only a small piece of a much larger asteroid, that is approximately the same size and mass as the first mission type.

With the basic list of mission requirements determined, a method to formulating the problem must be determined. In the next section, the basics of the problem formulation and objective function are outlined. The objective and constraint functions are then used in the optimization process with the proposed hybrid genetic-nonlinear programming algorithm (GNLP). The proposed hybrid GNLP algorithm is used to perform the optimization and will also be discussed prior to presenting the final results of the study.

7.2 Low-Thrust Problem Formulation

Low-thrust trajectory optimization methods typically fall into two categories, indirect and direct methods. Indirect methods are formulated with the calculus of variations, typically by creating a two-point boundary value problem, which provide an exact solution to the problem. Indirect solutions have the main advantage of low dimensionality and extremely accurate results. The disadvantages of indirect methods are that the formulation is particularly sensitive to the initial guess for both the physical states and the non-physical Lagrange multipliers (costates).

Direct methods parameterize the low-thrust trajectory problem. This formulation results in a much larger design space, often requiring large-scale nonlinear programming techniques to be used. While direct methods require a sufficiently accurate initial guess, the nonphysical Lagrange multiplies are not utilized, thus only an initial guess for physical parameters is required. The main drawback of direct methods is that they produce results with limited accuracy. In this paper, a new hybrid algorithm is presented to robustly calculate optimal low-thrust trajectories using a direct transcription model [40, 49, 89]. The problem of finding sufficiently accurate initial guesses is solved by utilizing a hybrid optimization algorithm, which incorporates a genetic algorithm along with non-linear programming solvers to perform a search to

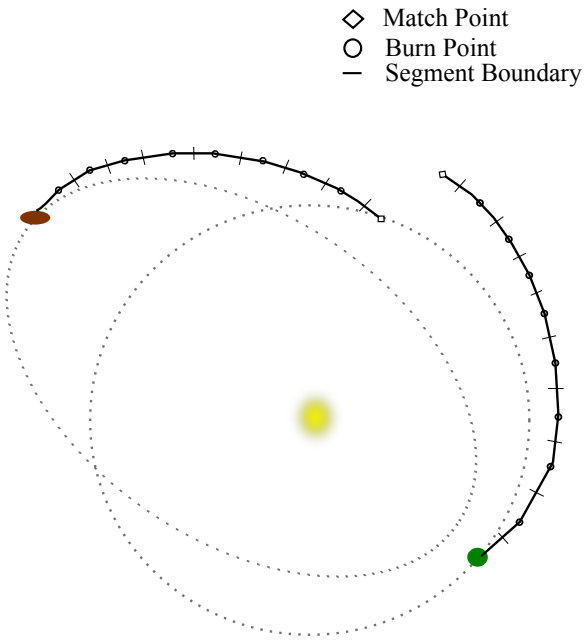


Figure 7.1 An Impulsive ΔV Low-Thrust Trajectory Model by Sims and Flanagan.

determine near globally optimal solutions. In this formulation, the randomly generated initial population for the genetic algorithm, are used as the initial guess for a non-linear programming (NLP) solver. The variables for each member of population are then updated with the results of the NLP solver. The typical genetic operators, crossover, mutation, reproduction, and survival of the fittest operators are then used to create a new population on the genetic operator. The process is repeated until the genetic algorithm converges on a solution.

7.2.1 Lambert Modified Sims-Flanagan Low-Thrust Model

Figure 7.1 illustrates the low-thrust trajectory model first described by Sims and Flanagan [89]. For each leg of the mission the low thrust trajectory arcs are modeled as a series of small impulsive ΔV maneuvers connected by conic 2-body arcs. Each leg of the trajectory is broken into N segments, with an impulse ΔV applied at the mid-point of each section.

The ΔV for each segment is not allowed to exceed a maximum magnitude, denoted by ΔV_{max-i}. This maximum impulsive ΔV is determined so that it cannot exceed the capability of the two Busek BHT-20K solar electric thrusters at full power. The maximum ΔV for each

segment can then be determined as a function of the maximum thrust and mass flow rate of the thrusters as follows:

$$\Delta V_{max,i} = \frac{T_{max}}{\dot{m}} \ln \frac{m_i}{m_i - \dot{m}\Delta t} \tag{7.1}$$

where the maximum time for each segment, Δt, is defined as

$$\Delta t = \frac{t_{leg}}{N} \tag{7.2}$$

In low-thrust optimization problems, the objective is often to maximize the final spacecraft mass. It is therefore necessary to update the spacecraft mass after each ΔV is applied. For this purpose, Tsiolkovsky's rocket equation is used to determine updated masses and is expressed as

$$m_{i+1} = m_i e^{-\frac{-\Delta V_i}{g_0 I_{sp}}} \tag{7.3}$$

For each leg of the mission, the trajectory is propagated, via solutions to Kepler's problem, forward from the starting point and backward from the ending point to a match point. The forward and backward propagated half legs are required to meet at the match point, which is typically ensured with six non-linear equality constraints. The match point is defined as follows:

$$\mathbf{Z}_{fw} - \mathbf{Z}_{bw} = [\Delta r_x, \Delta r_y, \Delta r_z, \Delta V_x, \Delta V_y, \Delta V_z]^T \tag{7.4}$$

The standard Sims-Flanagan transcription can be modified, in order to remove the need for the six equality constraints, by utilizing a solution to Lambert's problem to eliminate the match point equality conditions. Solutions to Lambert's problem can be used to determine the trajectory between two radius vectors and a given time-of-flight [7, 9, 10, 12, 13]. An illustration of the modified Sims-Flanagan transcription is shown in Fig. 7.2.

With this new formulation a velocity discontinuity will occur at both end points of the Lambert arc segment. These velocity discontinuities are then subject to two inequality constraints, in order to ensure the two velocities discontinuities don't exceed the capabilities of the solar

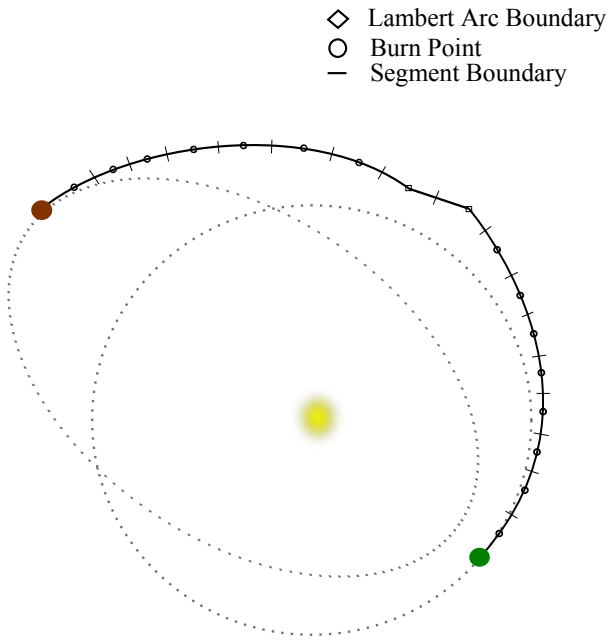


Figure 7.2 An Impulsive ΔV Low-Thrust Trajectory Model by Sims and Flanagan.

electric thrusters. The advantage to this modification is that six equality constraints, which can be difficult for NLP solvers to enforce, are replace with only two inequality constraints. The two inequality constraints are represented as:

$$g_1 = \Delta V_{fw} - \Delta V_{max} \tag{7.5}$$

$$g_2 = \Delta V_{bw} - \Delta V_{max} \tag{7.6}$$

The Earth departure velocity is determined as a function of the Earth escape C_3, provided by the lunar gravity-assist, and the departure ascension and declination angles. These two angles provide the direction of the Earth-system departure. The final Earth departure velocity is given as

$$\vec{V}_{dep} = \sqrt{c_3}\left[\cos\alpha\cos\beta\vec{I} + \sin\alpha\cos\beta\vec{J} + \sin\beta\vec{K}\right] \tag{7.7}$$

Sphere point picking is then used to determine the direction of each impulse maneuver. The direction of the ΔV is determined by the two spherical pointing angles, θ and ϕ. The

clock angle, θ has a range from 0 to 360 degrees, while the cone angle, ϕ, ranged from 0 to 180 degrees.

A throttle parameter, ϵ, which ranges from 0 to 1, is used to determined the magnitude of the ΔV applied at each burn point. This allows a ΔV range from 0 up to the maximum possible ΔV for each segment and is defined as

$$\Delta V_i = \epsilon \Delta V_{max,i} \tag{7.8}$$

The final ΔV vector is calculated from the two spherical angles and ΔV magnitude as follows:

$$\Delta \vec{V}_i = \Delta V_i \left[\cos\theta \sin\phi \vec{I} + \sin\theta \sin\phi \vec{J} + \cos\phi \vec{K} \right] \tag{7.9}$$

For the low-thrust transcription method used in this paper, using the spherical coordinates pose some problems. If the ΔV magnitude for any of the individual burns is 0, the two angle, θ and ϕ become meaningless. If this happens, the optimizer may make large changes to these angle which have no effect on the solution. When the ΔV is turned back on it could potentially be pointed in the wrong direction. This problem is minimized by limiting the minimum throttling parameter to a small non zero value. In this algorithm a minimum of 1e-5 is used, which allows the individual ΔV values (and the actual thrust) to be close to, but not exactly zero.

Another problem with the standard spherical coordinates occurs when the two pointing angles are near their bounds. If the cone angle, ϕ, is exactly 0 or 180 degrees θ becomes meaningless. As with the zero ΔV problem the optimizer will make large unnecessary changes to θ. This problem is alleviated by limiting ϕ to a range of 2 to 178 degrees. Problems with the clock angle, θ can also occur when the optimizer approaches a value close to the lower angle bound of 0 degrees. The optimal solution may actually be a negative angle, which is the same as a large position angle. If this happens the optimizer will be unable to make the jump towards the upper limit of 360 degree. This problem is solved by changing the clock angle limits form 0 to 360 degrees to -360 to 360 degrees. The variable limits for the spherical Sims-Flanagan transcription used with the GNLP optimization algorithm are shown in Table 7.2.

Table 7.2 Variable limits for the spherical Sims-Flanagan transcription model.

	α	β	θ_i	ϕ_i	ϵ_i
U_b	360	89	360	178	1
L_b	-360	-89	-360	2	1.E-05

7.2.2 ARM Design Problem Formulation

With the groundwork formulated for the Sims-Flanagan low-thrust problem, the next step is to develop an objective function for the ARM mission to be used with the hybrid GNLP solver. The hybrid solver is used to determine the initial launch date, time-of-flights, Earth departure and arrival variables, and the 3 spherical burn variables for each ΔV for both the departure and return tow phases. The Earth arrival velocity vector is determined in the same fashion as the Earth departure velocity.

The desired initial and final desired state variables for each leg of the mission are defined by the problem as functions of launch and final arrival times. In additional to the launch date, the initial departure state is also taken to be a function of launch C_3, as well as the launch ascension and declination, α and β (the final Earth arrival state vector is determined in the same manner), respectively as

$$\mathbf{Z}_{Earth-Dep} = \mathbf{f}\left(t_{Earth-Dep}, C_3, \alpha, \beta\right) \tag{7.10}$$

Because the GNLP solver is a constrained minimization algorithm, all nonlinear constraints are directly computed as constraint functions, which are evaluated the same time as the objective function. There are two inequality constraints from the Lambert solutions for both Earth departure and asteroid tow phase, which are defined as follow:

$$g_1 = \Delta V_{fw-dep} - \Delta V_{max-dep} \tag{7.11}$$

$$g_2 = \Delta V_{bw-dep} - \Delta V_{max-dep} \tag{7.12}$$

$$g_3 = \Delta V_{fw-arr} - \Delta V_{max-arr} \tag{7.13}$$

$$g_4 = \Delta V_{bw-arr} - \Delta V_{max-arr} \tag{7.14}$$

The final objective function, C, is constructed to maximize the final Earth arrival mass, which is done by minimizing the total change as

$$C = m_0 + m_{ast} - m_f \tag{7.15}$$

7.3 ARM Design Results

The objective of this study is to determine an algorithm that can be utilized for automated searches of asteroids. As previously shown, the objective function is formulated in such a way as to automate the choice of decision variables such as launch dates, times of flights, and all other mission parameters. With this approach asteroid redirect mission trajectories can be calculated for any asteroid. However, feasible trajectories can not always be determined. The results presented here are the asteroids where feasible mission were found.

For this study, a total of three separate asteroid return masses are tested. Each candidate asteroid is first assumed to have a retrievable mass of 500 metric tons. If a feasible trajectory is not found a mass of 250 metric tons is then tested, followed by a mass to 100 metric tons. Varying the retrievable mass, in order to determine feasible trajectories, allows for a minimum retrievable mass to be determined. A total of 17 possible asteroids were determined with feasible trajectories that have a retrievable mass of atleast 100 tons. With the exception of asteroid 2000 SG_{344} all of the asteroids have an H magnitude in the 27-29 range, corresponding to diameters in the 5-10 m range [92]. These asteroids can be towed by the spacecraft to the Earth vicinity. The asteroid 2000 SG_{344} has an estimated diameter in the 35-40 m range, for this mission it is assumed that an approximately 500 metric tons "boulder" would be collected and returned to the Earth.

The parameters for the spacecraft in this study are shown in Table 7.3. The mass margins for the spacecraft are taken from the Keck study [90] and the solar electric propulsion thruster information is that of the Busek BHT-20K [91]. For the missions shown below the required

Table 7.3 Initial Earth-Departure Spacecraft Parameters.

Thrust Power, kW	40
Specific Impulse, s	2,750
Number of Thrusters	2
Efficiency	70%
S/C Dry Mass, kg	5,500
S/C Mass at Earth Departure, kg	15,000

propellant can be no more than 9,500 kg. The missions typically remain close to 1 AU from the sun, so it is assumed that the thrusters can always operate at maximum thrust throughout the mission (the maximum thrust is typically not required for extended periods of the missions).

The results for the 17 possible target asteroids are shown in Table 7.4. The launch dates listed are the nominal launch dates found for each asteroid. In general the launch dates and time-of-flights can be adjusted for earlier and later launch dates with minimal penalties to the final Earth arrival mass. The nominal launch date for asteroid 2007 UN_{12} is in June of 2014, however, launch dates in the 2016 range are possible. Typical optimal time-of-flights range from 6-9 years. However, 5 of the possible candidate targets require time of flights in the 12-15 year range.

7.3.1 Detailed Example Missions

Detailed results for the top 4 candidate asteroids are shown in Table 7.5. All 4 of these mission have launch dates ranging from 2017 to 2021 and total time-of-flights from approximately 6-9 years. Each of these trajectories can return up to a 500 metric ton asteroid before 2023 to 2029, which is near the desired time frame for a human cis-lunar near-Earth asteroid (NEA) mission [90]. All of these mission require a retrieval ΔV under 200 m/s, which is crucial to minimizing the amount of fuel require for the mission.

The first asteroid examined is asteroid 2008 HU_4, which is the reference asteroid considered for the original Keck study [90]. The initial Earth departure spacecraft parameters for all of the mission are shown in Table 7.3. This mission is the first type of mission where the entire asteroid would be towed back to the Earth vicinity.

Table 7.4 Possible candidate for ARM missions

Asteroid	Launch Date	TOF	H (mag)	Maximum Returnable Asteroid Mass
2007 UN_{12}	24-Jun-14	6.3	28.7	500 t
2014 BA_3	1-Apr-15	8.8	28.2	250 t
2010 UE_{51}	25-Dec-17	5.9	28.3	500 t
2011 AA_{37}	25-May-18	11.6	22.8	100 t
2008 JL_{24}	22-Sep-18	6.5	29.6	250 t
2008 HU_4	11-Dec-18	7.4	28.2	500 t
2009 BD	12-Mar-19	15.2	28.1	500 t
2013 EC_{20}	2-May-19	14.0	29.0	250 t
2012 TF_{79}	30-Nov-19	8.8	27.4	250 t
2008 UA_{202}	2-Jul-20	8.2	29.4	250 t
2006 RH_{120}	15-Jul-20	9.3	29.5	500 t
2000 SG_{344}	3-Feb-21	7.3	24.7	500 t
2013 GH_{66}	6-Feb-21	7.2	28.0	100 t
2004 QA_{22}	25-Feb-21	7.7	27.9	100 t
2012 LA	9-Feb-22	12.8	27.6	250 t
2011 MD	10-Sep-22	14.8	28.0	500 t
2011 BL_{45}	5-Feb-23	7.5	27.0	250 t

The mission to 2008 HU_4 has the second earliest launch date out of 4 candidate asteroids, as well as a low Heliocentric ΔV for the return retrieval transfer to Earth at approximately 180 m/s. With an optimal launch date of Dec. 11th, 2018 this mission may be too early to complete the development of the spacecraft and develop the technology to capture the asteroid. It should be noted, as with the Keck study, that the launch date can be pushed back with minimal impact to the final mass returned to the Earth. This is due to the efficiency of the solar electric propulsion system and the low mass, relative to the return tow, during the initial heliocentric phase. The optimal return mass found for asteroid 2008 HU_4 is approximately 510 metric tons, leaving 5,500 kg of xenon propellant left for Earth proximity operations. Further details of this mission design can be found in Table 7.5.

The trajectory for the 2008 HU_4 retrieval mission is shown in Fig. 7.3. The initial Earth departure trajectory completes approximately 2.5 revolutions around the sun prior to arriving at the asteroid, stays at the asteroid for 117 days. The final retrieval tow phase of the trajectory completes approximately approximately 3.5 revolutions around the sun before arriving at the

Table 7.5 ARM mission design summer for the top four asteroids found in this study.

	2008 HU_4	**2006 RH_{120}**	**2000 SG_{344}**	**2010 EU_{51}**
Transfer to NEA				
Earth Departure Date	11-Dec-18	15-Jul-20	3-Feb-21	25-Dec-17
Escape C_3 (km^2/s^2)	2.0	2.0	2.0	2.0
Flight Time (days)	1239	1398	1227	1108
Heliocentric ΔV (km/s)	3.203	1.766	1.669	2.792
Asteroid Arrival Mass (kg)	13,320	14,049	14,100	13,525
Asteroid Arrival Date	11-Jan-22	13-May-24	13-Jun-24	5-Jan-21
Asteroid Mass (kg)	500,000	500,000	500,000	500,000
Transfer to Earth Vicinity				
Asteroid Stay Time	117	178	211	150
Departure Date	8-May-22	7-Noc-24	10-Jan-25	4-Jun-21
Departure Mass (kg)	513,320	514,049	514,100	513,525
Flight Time (days)	1450	1822	1217	881
Heliocentric Delta-V (km/s)	0.179	0.197	0.046	0.160
Arrival Mass (kg)	509,919	510,310	513,219	510,489
Arrival C_3 (km^2/s^2)	1.60	0.27	1.85	1.05
Earth Arrival Date	27-Apr-26	3-Nov-29	10-May-28	3-Nov-23
Total Xenon Used (kg)	**5081**	**4690**	**1781**	**4511**
Remaining Propellant (kg)	4419	4810	7719	4989
Total ΔV (km/s)	**3.382**	**1.963**	**1.716**	**2.952**
Total Flight Time (yrs)	**7.4**	**9.3**	**7.3**	**5.9**

Earth on April 27th, 2026 with a C_3 of 1.6 km^2/s^2

The thrust profile for the 2008 HU_4 mission is shown in Fig. 7.4. As the figure shows the maximum thrust is only necessary during the initial phase of the asteroid retrieval return leg. In addition to this, high thrust for the initial transfer phase is only require immediately after launch and immediately prior to arriving at the asteroid. There is an extended period with no thrust required for both transfer phases of the mission.

The second candidate trajectory and thrust profile shown is the mission to asteroid 2000 SG_{344}. This mission is the alternative mission type, in which pieces of a larger asteroid, are return to the Earth. The details for mission asteroid 2000 SG_{344} are provided in Table 7.5. The mission to asteroid 2000 SG_{344} has an Earth departure date of Feb. 3rd, 2021, with a total flight time of approximately 7.3 years, and a return mass of 513 metric tons. This mission has the highest propellant mass left for Earth proximity operations, at 7700 kg. This mission

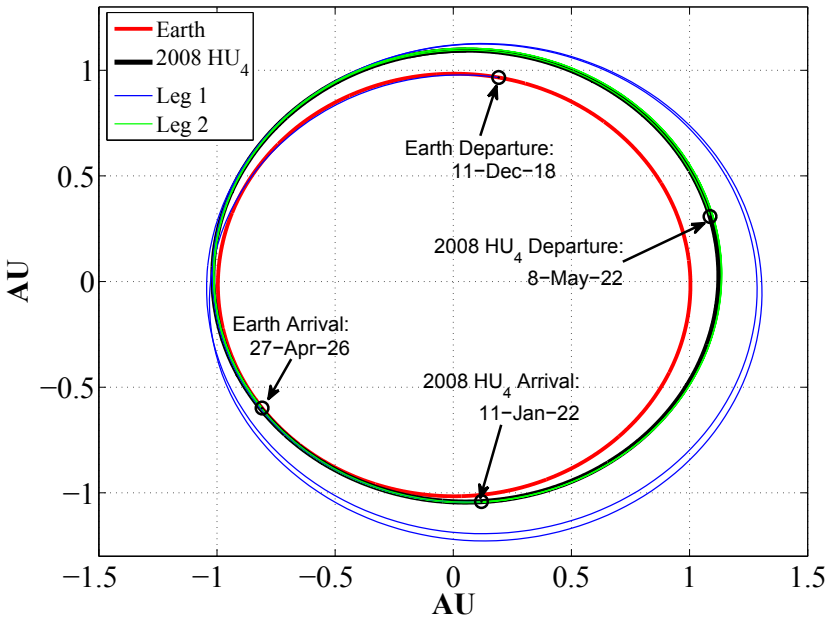


Figure 7.3 Trajectory for asteroid 2008 HU_4.

has the lowest propellant requirement at approximately 1800 kg. This mission has the lowest propellant requirement because the ΔV required for the retrieval phase is approximately 50 m/s, which is significantly lower than the return ΔV necessary for the other top 3 missions. This mission also has the highest arrival C_3, of the top 4 candidate asteroid, at approximately 1.85 km^2/s^2.

The trajectory for this mission is shown in Fig. 7.5. As this figure shows, asteroid 2000 SG_{344} has an orbit very close to that of the Earth. Both phases of this trajectory complete just under 3.5 revolutions around the sun. The final thrust profile for this trajectory is shown in Fig. 7.6. For this mission a thrust over 1.5 N is never required. This initial transfer to the asteroid requires thrusting during nearly the entire transfer, but never exceeds approximately 0.6 N. The return phase has one period of high thrust, just under 1.5 N, and an extended periods of coasting and thrusting during each revolution of the transfer.

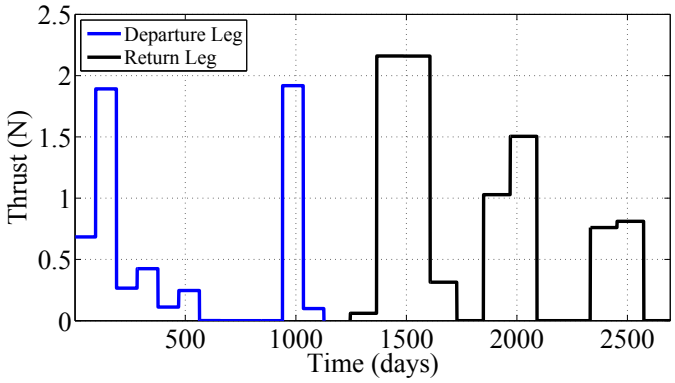


Figure 7.4 Thrust profile for asteroid 2008 HU_4.

7.4 Conclusion

A low-thrust mission trajectory design method has been implemented in conjunction with the new hybrid GNLP algorithm to determine feasible asteroid redirect trajectories. The algorithm has been developed to automate the search for possible target asteroids. A total of 17 possible target asteroids have been identified and presented in this paper. It has been shown that multiple approximately 500 ton asteroids can be retrieved with a 15 ton spacecraft. The top 4 asteroid candidates, based on maximum possible return mass, time-of-flight, and nominal launch dates are asteroids 2000 SG_{344}, 2006 RH_{120}, 2008 HU_4, and 2010 UE_{51}. With typical trajectories, several tons of propellant are left for Earth proximity operations, which could be utilized to establish stable orbits and/or for station keeping maneuvers.

In addition to identifying possible asteroid candidates an modified version of the original Sims-Flanagan transcription has been developed. This modification is able efficiently determine low thrust trajectories by utilizing solutions to Lambert's problem to replace the required 6 equality constraints with 2 inequality constraints. This modification allows the GNLP solver quickly determine feasible trajectory, with no position discontinuities, and enables the nonlinear programming solver to efficiently determine locally optimal trajectories.

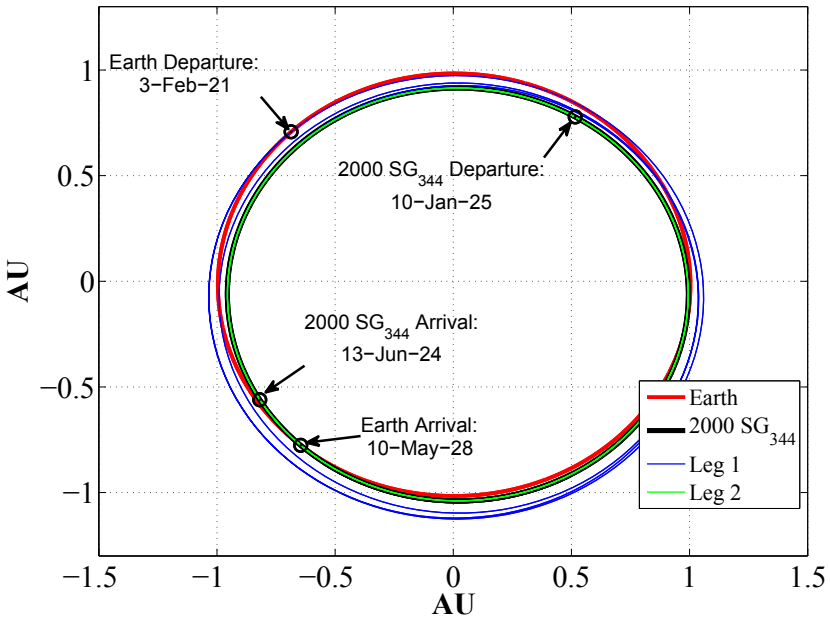


Figure 7.5 Trajectory for asteroid 2000 SG_{344}.

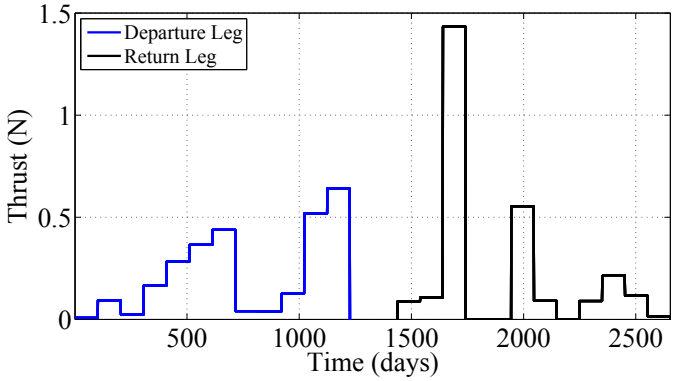


Figure 7.6 Thrust profile for asteroid 2000 SG_{344}.

CHAPTER 8. CONCLUSIONS

8.1 General Summary

This dissertation has discussed a variety of space mission analysis and design problems, for both asteroid deflection missions and planetary exploration missions. The mission types that have been optimized with the hybrid genetic algorithm-nonlinear programming (GNLP) include multiple gravity assist missions, with and without deep-space maneuvers, single and dual spacecraft asteroid intercept missions, and low-thrust asteroid redirect missions (ARM).

Mission analysis and design has historically been a time-consuming process with entire teams working to determine optimal trajectories for any given mission. When considering trajectories for missions such as NASA's Galileo, Cassini, or Messenger missions, the possible number of flyby trajectory sequences number in the tens of millions. Without efficient software to confidentially automate this process optimal trajectories and flyby sequences may easily be overlooked by mission designers.

The purpose of this work was to develop an optimization algorithm that can automate this design process for multiple gravity-assist (MGA), multiple gravity-assist with deep-space maneuvers (MGA-DSM), and low-thrust missions. The GNLP algorithm has been used to automate the process of determining complex trajectories, such as the Cassini mission, with no prior knowledge of the solution structure.

Other optimization algorithms developed for these type of missions often used nested evolutionary algorithms. A genetic algorithm is typically used for the outer loop to optimize the flyby order, while separate algorithm(s) are used to optimize the variables for individual trajectories [39, 40]. This inner loop solver often optimizes trajectories through the use of a cooperative algorithm, typically by alternating between particle swarm, differential evolution, or genetic

algorithms. Other research groups have focused on using advanced genetic algorithms [36–38] as their primary optimization algorithm. These types of algorithms often have long run times (typically 1+ days) and they often only find near optimal trajectories. The GNLP algorithm is able to directly optimize the number of gravity-assists, planetary flyby order, and mission design variables, resulting in a computationally efficient algorithm.

The GNLP algorithm has been shown to efficiently determine optimal trajectories. With run times to reproduce the Cassini mission, with no prior knowledge, of approximately 6 hours, the GNLP algorithm is very computationally efficient. The results presented in this dissertation represent an improvement in global optimization algorithms for both MGA and MGA-DSM trajectories, when compared to optimization algorithms commonly used for these types of missions [39, 40].

In addition to optimizing MGA and MGA-DSM type missions, the GNLP algorithm has also been used to efficiently optimize low-thrust asteroid redirect missions (ARM) utilizing a modified Sim-Flanagan transcription. This modification uses a solution to Lambert's problem to replace the midpoint equality constraints, for the position and velocity vector, with two inequality constraints. By replacing the 6 equality constraints with 2 inequality constraints the GNLP algorithm is able to efficiently determine possible candidates for the ARM mission. This process can be automated, in order to assess the feasibility of of new asteroids for ARM missions as they're discovered or when estimates for an asteroid's orbital are updated.

APPENDIX A. FORTRAN CODE FOR THE ORBITAL FUNCTIONS

This appendix contains the code used for many of the astrodynamic algorithms. The code is provided as is with no guarantees. If any of the codes or derivative works from codes in these appendices are used in another project the author simply asks that this dissertation be referenced to in any published works.

All of the included functions and subroutines have been tested to work with the GCC GFortran and Intel Fortran compilers. When used with the PGI or other Fortran compilers use of the intrinsic ISNAN function will likely cause problems.

Kepler's Solution Algorithms

The algorithm for solutions to Kepler's problem presented here are based on work presented in [14]. In the worst case scenario, the final algorithms converges within 8 iterations.

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
FUNCTION KEPLER(e,M_DUM)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: e,M_DUM
DOUBLE PRECISION :: KEPLER, TOL, ERROR, ENEW, EA, H, HNEW, M
INTEGER :: ITER, ITERMAX

TOL=1.d-8
ERROR=1.d0
ITERMAX=100
ITER=0
M=M_DUM

IF (e.LT.1.d0) THEN
    EA=M+e*SIN(M)+e**2/2.d0*SIN(2.d0*M)
    DO WHILE (ERROR.GE.TOL .AND. ITER.LE.ITERMAX)
```

```
        ENEW=EA+(M-EA+e*SIN(EA))/(1.d0-e*COS(EA))
        ERROR=ABS(EA-ENEW)
        EA=ENEW
        ITER=ITER+1
    END DO

    IF (ITER.GE.ITERMAX) THEN
        KEPLER=1.D12
    ELSE
        KEPLER=EA
    END IF

ELSE IF (e>=1.d0) THEN
    IF (e<1.6d0)THEN
        IF(M.LT.0.D0 .AND. M.GT.-PI)THEN
            H=M-e
        ELSEIF(M.GT.PI)THEN
            H=M-e
        ELSE
            H=M+e
        END IF
    ELSE
        IF(e.LT.3.6D0 .AND. ABS(M).GT.PI)THEN
            H=M-M/ABS(M)*e
        ELSE
            H=M/(e-1.D0)
        END IF
    END IF
    DO WHILE(ERROR.GE.TOL .AND. ITER.LE.ITERMAX)
        HNEW=H+(M-e*SINH(H)+H)/(e*COSH(H)-1.D0)
        ERROR=ABS(H-HNEW)
        H=HNEW
        ITER=ITER+1
    END DO

    IF (ITER.GE.ITERMAX) THEN
        KEPLER=1.D12
    ELSE
        KEPLER=H
    END IF
END IF

END FUNCTION KEPLER
```

Lambert Solution Algorithm

Battin Lambert Solution Algorithm

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS SUBROUTINE CONTAINS THE BATTIN LAMBERT SOLUTION METHOD, AS
!  DESCRIBED IN HIS BOOK.  IF A SOLUTION ISN'T CONVERGED UPON THEN
!  EACH MEMBER OF THE OUTPUT VELOCITY VECTORS ARE SET TO A VALUE OF
!  1.D12.  THE SUN GRAVITATIONAL CONSTANT IS ASSUMED, BUT THAT CAN
!  BE CHANGED BY MODIFYING THE CONSTANT MU PARAMETER.
!
!    REFERENCE:
!      TITLE: AN INTRODUCTION TO THE MATHEMATICS AND METHODS OF
!             ASTRODYNAMICS, REVISED EDITION
!      AUTHOR: RICHARD H. BATTIN
!      YEAR: 1999
!
!    INPUTS:
!      R1(3) = INITIAL RADIUS VECTOR (KM), DOUBLE PRECISION
!      R2(3) = FINAL RADIUS VECTOR (KM), DOUBLE PRECISION
!      DT    = REQUIRED TIME-OF-FLIGHT (SEC), DOUBLE PRECISION
!      OT    = ORBIT TYPE, 1 FOR PROGRADE, 2 FOR RETROGRADE, INTEGER
!
!    OUTPUTS:
!      V(6) = INITIAL AND FINAL VELOCITIES OF THE DETERMINED SOLUTION

SUBROUTINE LAMBERT_BATTIN(R1,R2,DT,OT, V)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: R1(3), R2(3), DT
DOUBLE PRECISION, INTENT(INOUT) :: V(6)
INTEGER, INTENT(IN) :: OT
DOUBLE PRECISION :: R1MAG, R2MAG,C12(3), TOL, NU, C, S, EPS, LTOP
DOUBLE PRECISION :: TANSQ2W, M, a, X, V1(3), V2(3), Y,Y1
DOUBLE PRECISION :: XNEW, dX, ROP, L, ETA
DOUBLE PRECISION :: DENOM, H1, H2, B, U,K, XI, FG(3), LAM,T_P,T
INTEGER :: ITERMAX, ITER

! CONVERGENCE TOLERANCE
TOL=1.d-8

R1MAG=NORM(R1)
R2MAG=NORM(R2)

! DETERMINE TRUE ANOMALY ANGLE HERE
```

```
C12=CROSS_PRODUCT(R1,R2)
NU=ACOS(DOT_PRODUCT(R1,R2)/(R1MAG*R2MAG))

!DETERMINE THE TRUE ANOMALY ANGLE USING THE ORBIT TYPE
!1 IS PROGRADE, 2 IS RETROGRADE
IF (OT==1) THEN
    IF (C12(3)<=0.d0) NU=2.d0*PI-NU
END IF

IF (OT==2) THEN
    IF (C12(3)>=0.d0) NU=2.d0*PI-NU
END IF

C=SQRT(R1MAG*R1MAG+R2MAG*R2MAG-2*R1MAG*R2MAG*COS(NU))
S=(R1MAG+R2MAG+C)/2.d0
EPS=(R2MAG-R1MAG)/R1MAG

LAM=SQRT(R1MAG*R2MAG)*COS(NU*.5D0)/S
T=SQRT(8.D0*MU/S**3)*DT
T_P=4.D0/3.D0*(1.D0-LAM**3)
M=T**2/(1.D0+LAM)**6

TANSQ2W=(EPS*EPS*0.25d0)/(SQRT(R2MAG/R1MAG)+R2MAG/R1MAG*&
        (2.d0+SQRT(R2MAG/R1MAG)))
ROP=SQRT(R2MAG*R1MAG)*(COS(NU*0.25d0)*COS(NU*0.25d0)+tansq2w)

IF (NU<PI) THEN
    LTOP=(SIN(NU*25.d-2)*SIN(NU*25.d-2)+TANSQ2W)
    L=LTOP/(LTOP+COS(NU*5.d-1))
ELSE
    LTOP=COS(NU*25.d-2)*COS(NU*25.d-2)+TANSQ2W
    L=(LTOP-COS(NU*5.d-1))/LTOP
END IF

!INITIAL GUESS IS SET HERE
IF(T.LE.T_P)THEN
    X=0.D0
    XNEW=0.D0
ELSE
    X=L
    XNEW=L
END IF

DX=1.d0
ITER=0
ITERMAX=20
! THIS TO LOOP DOES THE SUCCESSIVE SUBSTITUTION
```

```
DO WHILE(DX>=TOL .and. ITER<=ITERMAX)
    XI=XI_BATTIN(X)
    DENOM=(1.d0+2.d0*X+L)*(4.d0*X+XI*(3.d0+X))
    H1=(L+X)**2*(1.d0+3.d0*X+XI)/DENOM
    H2=(M*(X-L+XI))/DENOM
    B=27.d0*H2*25.d-2/(1.d0+H1)**3
    U=B/(2.d0*(SQRT(1.d0+B)+1.d0))
    K=K_BATTIN(U)
    Y=(1.d0+H1)/3.d0*(2.d0+SQRT(1.d0+B)/(1.d0+2.d0*U*K*K))
    XNEW=SQRT(((1.d0-L)/2.d0)**2+M/(Y*Y))-(1.d0+L)/2.d0
    Y1=SQRT(M/(L+X)*(1.d0+X));
    DX=ABS(X-XNEW);
    X=XNEW
    ITER=ITER+1;
END DO


IF (ITER >= ITERMAX .or. isnan(x) ) THEN
    ! IF A SOLUTION ISN'T FOUND THE FINAL VELOCITIES ARE
    !  OUTPUT AS A LARGE NUMBER
    V1(1)=1.D12
    V1(2)=1.D12
    V1(3)=1.D12
    V2(1)=1.D12
    V2(2)=1.D12
    V2(3)=1.D12
ELSE
    a=MU*DT*DT/(16.d0*ROP*ROP*X*Y*Y)
    FG=FG_BATTIN(a,S,C,NU,DT,R1MAG,R2MAG)
    V1=(R2-FG(1)*R1)/FG(2)
    V2=(FG(3)*R2-R1)/FG(2)
END IF

V(1:3)=V1
V(4:6)=V2

END SUBROUTINE LAMBERT_BATTIN

!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS FUNCTION COMPUTES THE FIRST CONTINUED FRACTION FOR THE BATTIN
!  ALGORITHM.
!
!    INPUT:
!      X = CURRENT X ESTIMATE, DOUBLE PRECISION
!
```

```
!    OUTPUT:
!      XI_BATTIN = VALUE FOR THE CONTINUED FRACTION, DOUBLE PRECISION
FUNCTION XI_BATTIN(X)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: X
DOUBLE PRECISION :: C(20), ETA, XI_BATTIN

C=(/ &
0.25396825396825395D0, 0.2525252525252254D0, 0.25174825174825177D0, &
0.25128205128205128D0, 0.25098039215686274D0, 0.25077399380804954D0, &
0.25062656641604009D0, 0.25051759834368531D0, 0.25043478260869567D0, &
0.25037037037037035D0, 0.25031928480204341D0, 0.25027808676307006D0, &
0.25024437927663734D0, 0.25021645021645023D0, 0.25019305019305021D0, &
0.25017325017325015D0, 0.25015634771732331D0, 0.25014180374361883D0, &
0.25012919896640828D0, 0.25011820330969264D0  /)

ETA=X/(SQRT(1.d0+X)+1.d0)**2;

XI_BATTIN=8.d0*(SQRT(1.d0+X)+1.d0)/(3.d0+1.d0/ &
(5.d0+ETA+9.d0/7.d0*ETA/(1.d0+C(1)*ETA/&
(1.d0+C(2)*ETA/(1.d0+C(3)*ETA/(1.d0+C(4)*ETA/(1.d0+C(5)*ETA/&
(1.d0+C(6)*ETA/(1.d0+C(7)*ETA/(1.d0+C(8)*ETA/(1.d0+C(9)*ETA/&
(1.d0+C(10)*ETA/(1.d0+C(11)*ETA/(1.d0+C(12)*ETA/(1.d0+C(13)*ETA/&
(1.d0+C(14)*ETA/(1.d0+C(15)*ETA/(1.d0+C(16)*ETA/(1.d0+C(17)*ETA/&
(1.d0+C(18)*ETA/(1.d0+C(19)*ETA/(1.d0+C(20)*ETA))))))))))))))))))))
END FUNCTION XI_BATTIN

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS FUNCTION COMPUTED THE K CONTINUED FRACTION FOR THE BATTIN
!  LAMBERT SOLUTION ALGORITHM.
!
!    INPUT:
!      U = SAME AS THE U VARIABLE IN THE BATTIN DESCRIPTTION,
!          DOUBLE PRECISION
!
!    OUTPUT:
!      K_BATTIN = CONTINUED FRACTION VALUE, DOUBLE PRECISION
FUNCTION K_BATTIN(U)
IMPLICIT NONE

DOUBLE PRECISION, INTENT(IN) :: U
DOUBLE PRECISION :: K_BATTIN, D(21)

D=(/ &
0.33333333333333331D0, 0.14814814814814814D0, 0.29629629629629628D0, &
```

```
0.22222222222222221D0, 0.27160493827160492D0, 0.23344556677890010D0, &
0.26418026418026419D0, 0.23817663817663817D0, 0.26056644880174290D0, &
0.24079807361541108D0, 0.25842383737120578D0, 0.24246606855302508D0, &
0.25700483091787441D0, 0.24362139917695474D0, 0.25599545906059318D0, &
0.24446916326782844D0, 0.25524057782122300D0, 0.24511784511784512D0, &
0.25465465465465464D0, 0.24563024563024563D0, 0.25418664443054689D0/)

K_BATTIN= D(1)/(1.d0+D(2)*U/(1.d0+D(3)*U/(1.d0+D(4)*U/&
  (1.d0+D(5)*U/(1.d0+D(6)*U/(1.d0+D(7)*U/(1.d0+D(8)*U/&
  (1.d0+D(9)*U/(1.d0+D(10)*U/(1.d0+D(11)*U/(1.d0+D(12)*U/&
  (1.d0+D(13)*U/(1.d0+D(14)*U/(1.d0+D(15)*U/(1.d0+D(16)*U/&
  (1.d0+D(17)*U/(1.d0+D(18)*U/(1.d0+D(19)*U/(1.d0+D(20)*U/&
  (1.d0+D(21)*U))))))))))))))))))))
END FUNCTION K_BATTIN

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS FUNCTION COMPUTES THE LAGRANGE COEFFICIENT VALUES TO COMPUTE
!  THE FINAL 2 VELOCITY VECTORS.  THE SUN'S GRAVITATIONAL PARAMETER
!  IS SPECIFIED BY THE SET CONSTANT VALUE, MU.  THIS CAN BE CHANGED
!  FOR OTHER BODIES
!
!    INPUT:
!      a =
!      S = SEMIPARAMETER, DOUBLE PRECISION
!      C = CHORD, DOUBLE PRECISION
!      NU = TRUE ANOMALY ANGLE (RAD), DOUBLE PRECISION
!      T = SCALED TIME-OF-FLIGHT PARAMETER, DOUBLE PRECISION
!      R1 = INITIAL RADIUS MAGNITUDE (KM), DOUBLE PRECISION
!      R2 = FINAL RADIUS MAGNITUDE (KM), DOUBLE PRECISION
!
!    OUTPUT:
!      FG_BATTIN(1) = F, DOUBLE PRECISION
!      FG_BATTIN(2) = G, DOUBLE PRECISION
!      FG_BATTIN(3) = G_DOT, DOUBLE PRECISION
!
FUNCTION FG_BATTIN(A,S,C,NU,T,R1,R2)
IMPLICIT NONE

DOUBLE PRECISION, INTENT(IN) :: A, S, C, NU, T, R1, R2
DOUBLE PRECISION :: FG_BATTIN(3), SMALL_NUMBER, BE, A_MIN, T_MIN, &
    DUM, AE, DE, F, G, GDOT, AH, BH, DH

SMALL_NUMBER= 1.D-3

IF (A>SMALL_NUMBER) THEN
```

```
    BE=2.D0*ASIN((SQRT((S-C)/(2*A))))
    IF (NU>PI) BE=-BE

    A_MIN=S*5.D-1
    T_MIN=SQRT(A_MIN**3/MU)*(PI-BE+SIN(BE))
    DUM=(SQRT(S/(2.D0*A)))
    AE=2.D0*ASIN(DUM)
    IF (T>T_MIN) AE=2.D0*PI-AE

    DE=AE-BE
    F=1.D0-A/R1*(1-COS(DE))
    G=T-SQRT(A*A*A/MU)*(DE-SIN(DE))
    GDOT=1.D0-A/R2*(1.D0-COS(DE))

ELSE IF (A<-SMALL_NUMBER) THEN
    AH=2.D0*ASINH1(SQRT(S/(-2.D0*A)))
    BH=2.D0*ASINH1(SQRT((S-C)/(-2.D0*A)))
    DH=AH-BH
    F=1.D0-A/R1*(1.D0-COSH(DH))
    G=T-SQRT(-A**3/MU)*(SINH(DH)-DH)
    GDOT=1.D0-A/R2*(1.D0-COSH(DH))

ELSE
    F=0.D0
    G=0.D0
    GDOT=0.D0
END IF

FG_BATTIN(1)=F
FG_BATTIN(2)=G
FG_BATTIN(3)=GDOT

END FUNCTION FG_BATTIN
```

Gooding Lambert Solution Algorithm

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!  THIS SECTION CONTAINS THE SUBROUTINES FOR THE GOODING LAMBERT
!    LAMBERT SOLUTION
!
!    INPUTS:
!      R1 = INITIAL RADIUS VECTOR, DOUBLE PRECISION
!      R2 = FINAL RADIUS VECTOR, DOUBLE PRECISION
!      DT = TRANSFER TIME, MUST AGREE WITH MU, DOUBLE PRECISION
```

```
!      OT = ORBIT TYPE, 1=PROGRADE, 2=RETROGRADE, INTEGER
!
!    OUTPUTS:
!      V(1:3) = INITIAL VELOCITY VECTOR, DOUBLE PRECISION
!      V(4:6) = FINAL VELOCITY VECTORS, DOUBLE PRECISION
!
SUBROUTINE LAMBERT_GOODING(R1,R2,DT,OT, V)
IMPLICIT NONE

DOUBLE PRECISION, INTENT(IN) :: R1(3), R2(3), DT
DOUBLE PRECISION, INTENT(INOUT) :: V(6)
INTEGER, INTENT(IN) :: OT

DOUBLE PRECISION :: TOL, R1MAG, R2MAG, C12(3), NU, C12MAG, &
    TAN1U(3), TAN2U(3), C, S, T, Q, T0, R1UNIT(3), R2UNIT(3), &
    C12UNIT(3), C0, C1, C2, C3, X0, X01, X02, X03, W_BIG, &
    W_LITTLE, LAM, GAMMA, V1(3), V2(3), X, SIG, RHO, Z, &
    VR1(3), VR2(3), VT1(3), VT2(3), DUM, T2, TD1, TD2, TDIFF, &
    X_NEW
INTEGER :: SIGNX, ERROR1, ERROR2, ITER, ITERMAX, NREV
NREV=0
ITERMAX=20
TOL=1.D-8
ERROR1=0
ERROR2=0

R1MAG=NORM(R1)
R2MAG=NORM(R2)
R1UNIT=R1/R1MAG
R2UNIT=R2/R2MAG

C12=CROSS_PRODUCT(R1, R2)
C12MAG=NORM(C12)
C12UNIT=C12/C12MAG

NU=ACOS(DOT_PRODUCT(R1,R2)/(R1MAG*R2MAG))

!DETERMINE THE TRUE ANOMALY ANGLE USING THE ORBIT TYPE
!1 IS PROGRADE, 2 IS RETROGRADE
IF (OT==1) THEN
    IF (C12(3) <= 0.D0) THEN
        NU=2.D0*PI-NU;
    END IF
END IF
IF (OT==2) THEN
    IF(C12(3)>=0.D0) THEN
        NU=2.D0*PI-NU;
```

```
    END IF
END IF

!TANGENTIAL UNIT VECTORS
TAN1U=CROSS_PRODUCT(C12UNIT, R1UNIT)
TAN2U=CROSS_PRODUCT(C12UNIT, R2UNIT)
IF (NU >=PI) THEN
   TAN1U=-TAN1U
   TAN2U=-TAN2U
END IF

!============CONSTANTS DETERMINED FORM PROBLEM  GEOMETRY==================
C=SQRT(R1MAG*R1MAG+R2MAG*R2MAG - 2.D0*R1MAG*R2MAG*COS(NU))
S=(R1MAG+R2MAG+C)/2.D0
T=SQRT(8.D0*MU/S**3)*DT
Q=SQRT(R1MAG*R2MAG)/S * COS(NU*.5D0)
IF (NU<=PI) THEN
    Q=ABS(Q)
ELSE
    Q=-ABS(Q)
END IF

!INITIAL CONDITIONS
DUM=0.D0
CALL TIME_DERIVATIVES(DUM,Q,NREV, T0,DUM,DUM)

IF ((T0-T)<0.D0) THEN
    SIGNX=-1
ELSE
    SIGNX=1
END IF
!=================CONSTANTS FOR INITIAL CONDITIONS SPLICING===============
C0=1.7D0
C1=0.5D0
C2=0.03D0

!INITIAL VALUES FOR SINGLE REVOLUTION CASE
X0=0.D0

IF (SIGNX==1) THEN
    X0=T0*(T0-T)/(4.D0*T)
ELSE
    X01=-(T-T0)/(T-T0+4.D0);
    X02=-SQRT((T-T0)/(T+0.5D0*T0))
    W_BIG=X01+C1*SQRT(2.D0-NU/PI)
    IF (W_BIG>=0.D0) THEN
        X03=X01
```

```
    ELSE
        W_LITTLE=SQRT(SQRT(SQRT(SQRT(-W_BIG))))
        X03=X01+W_LITTLE*(X02-X01)
    END IF
        LAM=1.D0+C1*X03*(4.D0/(4.D0-T-T0))-C2*X03**2*SQRT(1.D0+X01)
        X0=LAM*X03
END IF

IF (X0<=-1) THEN
    ERROR1=1
END IF

!============SOLVE EACH CASE GIVEN THE INITIAL CONDITIONS ABOVE============
!FOR 0-REVOLUTION CASE THE VL'S ARE RETURNED AS NAN'S AND THE VH'S RETURN
!THE CALCULATED VELOCITIES
GAMMA=SQRT(MU*S/2.D0)
IF (ERROR1==1) THEN
    V1=1.D7
    V2=1.D7
ELSE

!X=HALLEY(X0,T,TOL,Q,NREV)
ITER=0
X=X0

DO WHILE (ABS(TDIFF)>=TOL .AND. ITER <=ITERMAX)
    ITER=ITER+1
    CALL TIME_DERIVATIVES( DUM, Q, NREV, T2, TD1, TD2)
    TDIFF=T2-T
    X_NEW=X-2.D0*TDIFF*TD1/(2.D0*TD1*TD1-TDIFF*TD2)
    X=X_NEW
END DO

IF (ITER.GE.ITERMAX .OR. ISNAN(X)) THEN
    V1=1.D12
    V2=1.D12
ELSE
    IF (C==0) THEN
        SIG=1.D0
        RHO=0.D0
        Z=ABS(X)
    ELSE
        SIG= 2.D0*SQRT(R1MAG*R2MAG/C**2)*SIN(.5D0*NU)
        RHO= (R1MAG-R2MAG)/C
        Z=   SQRT(1.D0+Q**2*(X**2-1.D0))
    END IF
    !RADIAL COMPONENTS OF VELOCITY
```

```
    DUM= GAMMA*((Q*Z-X)-RHO*(Q*Z+X))/R1MAG
    VR1=DUM*R1UNIT
    DUM=-GAMMA*((Q*Z-X)+RHO*(Q*Z+X))/R2MAG
    VR2=DUM*R2UNIT
    !TANGENTIAL COMPONENTS OF VELOCITY
    DUM=GAMMA*SIG*(Z+Q*X)
    VT1=DUM/R1MAG*TAN1U
    VT2=DUM/R2MAG*TAN2U
    V1=VR1+VT1
    V2=VR2+VT2
    END IF
END IF
V(1:3)=V1
V(4:6)=V2
END SUBROUTINE LAMBERT_GOODING

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS FUNCTION CALCULATES THE TIME-OF-FLIGHT EQUATIONS AND THE
!  DERIVATIVES
!
!    INPUTS:
!      X = CURRENT X ESTIMATE, DOUBLE PRECISION
!      Q = LAMBERT PARAMETER, DOUBLE PRECISION
!      NREV = NUMBER OF COMPLETE REVOLUTIONS, INTEGER
!
!    OUTPUTS:
!      T = CALCULATED TIME-OF-FLIGHT, DOUBLE PRECISION
!      TD1 = TIME-OF-FLIGHT FIRST DERIVATE, DOUBLE PRECISION
!      TD2 = TIME-OF-FLIGHT SECOND DERIVATIVE, DOUBLE PRECISION
!
SUBROUTINE TIME_DERIVATIVES(X, Q, NREV, T, TD1, TD2)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: Q
DOUBLE PRECISION, INTENT (IN OUT) :: T, TD1, TD2, X
INTEGER, INTENT(IN) :: NREV
DOUBLE PRECISION :: K, E, DELT, SIG1, DSIG1, D2SIG1, D3SIG1
DOUBLE PRECISION :: SIG2, DSIG2, D2SIG2, D3SIG2, Y, Z, F ,G, D,XX
DOUBLE PRECISION :: U, BETA, TOL

TOL=1.D-8
DELT=1.D-1
XX=X
IF (XX<-1.D0) THEN
    XX=ABS(X) - 2.D0
ELSEIF (X==-1) THEN
```

```
    XX=X+DELT
END IF

K=Q*Q
E=XX*XX-1.D0
IF (XX==1) THEN
    ! EXACT PARABOLIC SOLUTION, PROBABLY NEVER ACTUALLY USED
    T  = 4.D0/3.D0*(1-Q**3)
    TD1= 4.D0/5.D0*(Q**5 - 1.D0)
    TD2= TD1+120.D0/70.D0*(1.D0-Q**7)
ELSE IF (ABS(XX-1.D0)<TOL) THEN
    ! NEAR PARABOLIC SOLUTION, USE THE TRANS/SERIES REPRESENTATION
    CALL SIGMA( -E, SIG1, DSIG1, D2SIG1)!, D3SIG1)
    CALL SIGMA( -E*Q*Q, SIG2, DSIG2, D2SIG2)!, D3SIG2)
    T=SIG1- Q**3*SIG2
    TD1=2.D0*X*(Q**5.D0*DSIG2-DSIG1)
    TD2=TD1/X+4.D0*X*X*(D2SIG1-Q**7.D0*D2SIG2)

ELSE
    ! ALL CASES NOT EXACTLY PARABLOLIC OR CLOSE TO PARABOLIC
    Y=SQRT(ABS(E))
    Z=SQRT(1.D0-Q**2+Q**2*X**2)
    F=Y*(Z-Q*X)
    U=-E
    BETA=Q*Z-X
    G=(X**2-Q**2*(-E))/(X*Z-Q*(-E))
    IF (E<0) THEN
        D=ATAN2(F,G)+PI*DBLE(NREV)
    ELSE
        D=ATANH(F/G)
        !D=LOG(F+G)
    END IF

    T=2.D0*(X-Q*Z-D/Y)/E
    TD1=(3.D0*X*T+4.D0*Q**3*X/Z-4.D0)/U
    TD2=(3.D0*T+5.0*X*TD1+4.D0*(Q/Z)**3*(1.D0-Q**2))/U
END IF
END SUBROUTINE TIME_DERIVATIVES

!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS FUNCTION COMPUTES THE SIGMA AND SIGMA DERIVATIVES FOR THE
!  GOODING LAMBERT SOLUTION
!
!    INPUT:
```

```
!      U = INITIAL U VALUE FOR THE SIGMA FUNCTION, DOUBLE PRECISION
!
!    OUTPUT:
!      SIGM = SIGMA(U), DOUBLE PRECISION
!      DSIGMA = D(SIGMA)/DU, DOUBLE PRECISION
!      D2SIGMA = D2(SIGMA)/DU2, DOUBLE PRECISION
!
SUBROUTINE sigma(u, sigm, dsigma, d2sigma)
IMPLICIT NONE
DOUBLE PRECISION , INTENT(IN) :: U
DOUBLE PRECISION, INTENT(INOUT) :: SIGM, DSIGMA, D2SIGMA

SIGM=-2.D0*(SQRT(U)*SQRT(1.D0-U)-ASIN(SQRT(U)))/SQRT(U)**3
DSIGMA=-(U/SQRT(1.D0-U)+(3.D0*asin(SQRT(U)))/SQRT(U)-&
       3.D0/SQRT((1.D0-U)))/U**2
D2SIGMA=(15.D0*ASIN(SQRT(U))*SQRT(1.D0-U)**3-15.D0*SQRT(U)+&
        20.D0*SQRT(U)**3-3.D0*SQRT(U)**5 )/&
        (2.D0*SQRT(U)**7*SQRT((1.D0-U))**3)
END SUBROUTINE SIGMA
```

Sun Lambert Solution Algorithm

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS SUBROUTINE CONTAINS THE ALGORITHM FOR THE SUN LAMBERT
!  SOLUTION.
!
!    INPUTS:
!      R1(3) = INITIAL RADIUS VECTOR (KM), DOUBLE PRECISION
!      R2(3) = FINAL RADIUS VECTOR (KM), DOUBLE PRECISION
!      DT    = TIME-OF-FLIGHT (SEC), DOUBLE PRECISION
!      OT    = ORBIT TYPE, 1=PROGRADE, 2=RETROGRADE, INTEGER
!
!    OUTPUTS:
!      V(1:3) = INITIAL VELOCITY VECTOR (KM/S), DOUBLE PRECISION
!      V(4:6) = FINAL VELOCITY VECTOR (KM/S), DOUBLE PRECISION
SUBROUTINE lambert_sun(R1,R2,DT,OT, V)
IMPLICIT NONE
INTEGER, INTENT(IN) :: OT
DOUBLE PRECISION, INTENT(IN) :: R1(3), R2(3), DT
DOUBLE PRECISION, INTENT(INOUT) :: V(6)
DOUBLE PRECISION :: TOL, R1MAG, R2MAG, C12(3), NU, SIGMA, C, M, N, &
    TAU, TAU_P, TAU_ME, X, XNEW, DX, Y, F, FP, FPP, N_F, V1(3), &
    V2(3), VR, VC, EC(3), ER1(3), ER2(3), sign_fp
INTEGER :: ITERMAX, ITER
```

```
ITERMAX=20
TOL=1.D-8

R1MAG=NORM(R1)
R2MAG=NORM(R2)
C12=cross_product(R1, R2)
NU=ACOS(DOT_PRODUCT(R1,R2)/(r1mag*r2mag))

!Determine the true anomaly angle using the orbit type
!1 is prograde, 2 is retrograde
IF (OT==1) THEN
    IF(C12(3) <= 0.D0) THEN
        NU=2.D0*PI-NU;
   END IF
END IF

IF (OT==2) THEN
    IF(C12(3)>=0.D0) THEN
        NU=2.D0*PI-NU;
    END IF
END IF

C=SQRT(R1MAG*R1MAG+R2MAG*R2MAG-2*R1MAG*R2MAG*COS(NU))
M=R1MAG+R2MAG+C
N=R1MAG+R2MAG-C
SIGMA=ABS(SQRT(4.D0*R1MAG*R2MAG/M**2*COS(0.5D0*NU)**2))

IF(NU<PI-TOL) THEN
    SIGMA=SIGMA
ELSEIF (NU>PI+TOL) THEN
    SIGMA=-SIGMA
ELSE
    SIGMA=0
END IF

TAU=4.D0*DT*SQRT(MU/M**3)
TAU_P=2.D0/3.D0*(1.D0-SIGMA**3)


IF (TAU>TAU_P+TOL) THEN !ELLIPTICAL ORBIT
    !INITIAL VALUES FOR ELIPTICAL AND PARABOLIC ORBITS
    TAU_ME=ACOS(SIGMA)+SIGMA*SQRT(1.D0-SIGMA**2)
    IF(TAU<TAU_ME-TOL) THEN
        X=0.5D0
    ELSEIF (TAU>TAU_ME+TOL)THEN
        X=-0.5D0
    ELSE
```

```
        X=0.D0
    END IF
ELSEIF(TAU<TAU_P-TOL)THEN
    !HYPERBOLIC ORBIT
    X=1.5D0
ELSE
    !PARABOLIC ORBIT
    X=1.D0
END IF

DX=1.D0
ITER=0
N_F=4.D0

DO WHILE(DX>=TOL .AND. ITER.LE.ITERMAX)
    ITER=ITER+1
    CALL lambert_sun_F(X,SIGMA,F,FP,FPP,TAU,NU)
    SIGN_FP=-FP/ABS(FP)
    !LAGUERRE
    !XNEW=X-N_F*F/(FP+SIGN_FP*FP/ABS(FP)*SQRT(ABS((N_F-1.D0)**2*&
    !     FP**2-N_F*(N_F-1)*F*FPP)))

    !HALLEY
    XNEW=X-2.D0*F*FP/(2.D0*FP**2-F*FPP)
    DX=ABS(X-XNEW)
    X=XNEW
END DO

IF (ITER >= ITERMAX .or. isnan(x)) THEN
    V1(1)=1.D12; V1(2)=1.D12; V1(3)=1.D12
    V2(1)=1.D12; V2(2)=1.D12; V2(3)=1.D12
ELSE
    IF(NU<PI-TOL) THEN
        Y=SQRT(1.D0-SIGMA*SIGMA*(1.D0-X*X))
    ELSEIF (NU>PI+TOL)THEN
        Y=-SQRT(1.D0-SIGMA*SIGMA*(1.D0-X*X))
    ELSE
        Y=1
    END IF

    VC=SQRT(MU)*(Y/SQRT(N)+X/SQRT(M))
    VR=SQRT(MU)*(Y/SQRT(N)-X/SQRT(M))
    EC=(R2-R1)/C
    ER1=R1/R1MAG
    ER2=R2/R2MAG
    V1=VC*EC+VR*ER1
    V2=VC*EC-VR*ER2
```

```
END IF

V(1:3)=V1
V(4:6)=V2

END SUBROUTINE lambert_sun

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE CALCULATES THE TIME-OF-FLIGHT EQUATION AND THE
!  DERIVATIVES.
!
!    INPUTS:
!      X     = CURRENT X VALUE, DOUBLE PRECISION
!      SIGMA = LAMBERT PARAMETER, DOUBLE PRECISION
!      TAU   = SPECIFIED TIME PARAMETER, DOUBLE PRECISION
!      NU    = TRUE ANOMALY (RAD), DOUBLE PRECISION
!
!    OUTPUTS:
!      F   = TIME-OF-FLIGHT EQUATION, DOUBLE PRECISION
!      FP  = FIRST DER.E OF THE TIME EQUATION, DOUBLE PRECISION
!      FPP = SECOND DER. OF THE TIME EQUATION, DOUBLE PRECISION
!
SUBROUTINE lambert_sun_F(X, SIGMA, F, FP, FPP, TAU, NU)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: SIGMA, TAU, NU
DOUBLE PRECISION, INTENT(INOUT)  :: X, F, FP, FPP
DOUBLE PRECISION  :: TOL, DUM1,DUM2,Y, COTX, COTY
TOL=1.D-12

IF(NU<PI-TOL) THEN
    Y=SQRT(1.D0-SIGMA*SIGMA*(1.D0-X*X))
ELSEIF (NU>PI+TOL)THEN
    Y=-SQRT(1.D0-SIGMA*SIGMA*(1.D0-X*X))
ELSE
    Y=1.d0
END IF

IF (X.LE.(1.D0-TOL))THEN
    COTX=DBLE(ACOS(X))
    COTY=ATAN(SQRT(1.D0-Y*Y)/Y)
    F=1.D0/SQRT((1.D0-X*X)**3)*(COTX-COTY-X*SQRT(1.D0-X*X)+&
      Y*SQRT(1.D0-Y*Y))
ELSE IF (X.GE.(1.D0+TOL)) THEN
    DUM1=X/SQRT(X*X-1.D0)
    DUM2=Y/SQRT(Y*Y-1.D0)
```

```
    F=1.D0/SQRT((X*X-1.D0)**3)*(-ACOTH1(DUM1)+ACOTH1(DUM2)+&
      X*SQRT(X*X-1.D0)-Y*SQRT(Y*Y-1.D0))
ELSE
    F=2.D0/3.D0*(1.D0-SIGMA**3)
END IF

FP=1.D0/(1.D0-X*X)*(3.D0*X*F-2.D0*(1.D0-SIGMA**3*X/ABS(Y)))
FPP=1.D0/(X*(1.D0-X*X))*((1.D0+4.D0*X*X)*FP+2.D0*&
    (1.D0-SIGMA**5*X**3/ABS(Y)**3))
F=F-TAU
END SUBROUTINE lambert_sun_F
```

Universal variable Lambert Solution Method

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS FUNCTION CALCULATES THE C STUMPFF FUNCTION
!
!    INPUTS:
!      Z = CURRENT Z VARIABLE ESTIMATE, DOUBLE PRECISION
!
!    OUTPUTS:
!      C_STUMPFF = C STUMPFF FUNCTION VALUE, DOUBLE PRECISION
FUNCTION C_STUMPFF(Z)
IMPLICIT NONE

DOUBLE PRECISION :: C_STUMPFF,DUM, SMALL
DOUBLE PRECISION, INTENT(IN) :: Z

SMALL=1.d-6

    IF (Z>SMALL) THEN
        DUM=(1.d0-COS(SQRT(Z)))/Z
    ELSEIF (Z<-SMALL) THEN
        DUM=(COSH(SQRT(-Z))-1.d0)/(-Z)
    ELSE
        DUM=0.5d0
    END IF

    C_STUMPFF=DUM

END FUNCTION C_STUMPFF

!*****************************************************************!
!*****************************************************************!
```

```
!*****************************************************************!
!*****************************************************************!
! THIS FUNCTION CALCULATES THE S STUMPFF FUNCTION
!
!    INPUTS:
!      Z = CURRENT Z VARIABLE ESTIMATE, DOUBLE PRECISION
!
!    OUTPUTS:
!      S_STUMPFF = S STUMPFF FUNCTION VALUE, DOUBLE PRECISION
FUNCTION S_STUMPFF(Z)
IMPLICIT NONE

DOUBLE PRECISION :: S_STUMPFF, DUM, SMALL
DOUBLE PRECISION, INTENT(IN) :: Z
SMALL=1.d-6

IF(Z>SMALL) THEN
    DUM=(SQRT(Z)-SIN(SQRT(Z)))/SQRT(Z)**3
ELSEIF (Z<-SMALL) THEN
    DUM=(SINH(SQRT(-Z))-SQRT(-Z))/SQRT(-Z)**3
ELSE
    DUM=1.d0/6.d0
END IF

S_STUMPFF=DUM

END FUNCTION S_STUMPFF
```

Other Algorithms

This section contains the Fortran code for other important algorithms related to orbital mechanics. This includes, calculation of Julian Dates, conversions, generating planetary ephemeris data,

Julian Date Calculation

This algorithm converts a standard Gregorian date to the Julian date number. This algorithm was adapted from Vallado [14] and is valid from March 1^{st}, 1900 to Feb. 28^{th}, 2100. If an algorithm is needed to calculated the Julian date outside of this range the conversion algorithm from [93] can be substituted.

```
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
```

```
!*********************************************************************!
! THIS FUNCTION TAKES THE DATE, IN INTEGER VALUES AND OUTPUTS THE
!  JULIAN DATES.  THIS ALGORITHM IS VALID FROM MARCH 1, 1900 TO
!  FEB. 28, 2100.
!
!    INPUTS:
!      YR = YEAR, INTEGER
!      MO = MONTH, INTEGER
!      D  = DAY, INTEGER
!      M  = MONTH, INTEGER
!      S = SECOND, INTEGER
!
!    OUTPUT:
!      JDATE = JULIAN DATE, DOUBLE PRECISION
!
FUNCTION JDATE(YR,MO,D,H,M,S)
IMPLICIT NONE

DOUBLE PRECISION :: JDATE
INTEGER, INTENT(IN) :: yr,mo,d,h,m,s

JDATE=367.d0*DBLE(YR)-FLOOR(7.d0*(DBLE(YR)+FLOOR((DBLE(MO)+&
      9.d0)/12.d0))/4.d0) + FLOOR(275.d0*DBLE(MO)/9.d0)+DBLE(D)+ &
      1.7210135d6+(((DBLE(S)/60.d0+DBLE(M)))/60.d0+dble(H))/24.d0

END FUNCTION JDATE
```

Calculation of the State Vector from Orbital Elements

```
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!THIS ALGORITHM CONVERTS THE ORBITAL ELEMENTS TO THE STATE VECTOR
! INPUTS ARE ALL DOUBLE PRECISION ::
!     a = SEMI-MAJOR AXIS (KM)
!     e = ORBIT ECCENTRICITY
!     i = ORBIT ECCENTRICITY  (RAD)
!     RA= LONGITUDE OF THE ASCENDING NODE  (RAD)
!     w = ARGUMENT OF PERIAPSIS  (RAD)
!     nu= TRUE ANOMALY  (RAD)
! OUTPUT IS A DOUBLE PRECISION ARRAY WITH A LENGTH OF 6 ::
!     SV(1:3)=R
!     SV(4:6)=V
!
! THE SUNS GRAVITATIONAL PARAMETER(mu) IS DEFINED AS A CONSTANT.
! FOR OTHER BODIES THIS CAN BE CHANGED.
```

```
!
FUNCTION OE2SV(a,e,i,RA,w,NU)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(IN) :: a,e,i,RA,w,NU
DOUBLE PRECISION :: OE2SV(6), P,H,R, R_PQW(3),V_PQW(3), &
    R_VEC(3), V_VEC(3), RMAT(3,3)

P=a*(1.d0-e*e)
H=sqrt(P*MU)
R=P/(1.d0+e*COS(NU))

R_PQW=(/ R*COS(NU), R*SIN(NU), 0.d0 /)
V_PQW=(/ -SIN(NU), e+COS(NU), 0.d0 /)
V_PQW=MU/H*V_PQW

RMAT(1,1)=COS(RA)*COS(w)-SIN(RA)*SIN(w)*COS(i)
RMAT(1,2)=-COS(RA)*SIN(w)-SIN(RA)*COS(w)*COS(i)
RMAT(1,3)=SIN(RA)*SIN(i)
RMAT(2,1)=SIN(RA)*COS(w)+COS(RA)*SIN(w)*COS(i)
RMAT(2,2)=-SIN(RA)*SIN(w)+COS(RA)*COS(w)*COS(i)
RMAT(2,3)=-COS(RA)*SIN(i)
RMAT(3,1)=SIN(w)*SIN(i)
RMAT(3,2)=COS(w)*SIN(i)
RMAT(3,3)=COS(i)

R_VEC=MATMUL(RMAT,R_PQW)
V_VEC=MATMUL(RMAT,V_PQW)

OE2SV(1:3)=R_VEC
OE2SV(4:6)=V_VEC

END FUNCTION OE2SV
```

Calculation of the Orbital Elements from the State Vector

```
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!THIS ALGORITHM CONVERTS THE ORBITAL ELEMENTS TO THE STATE VECTOR
! INPUTS ARE ALL DOUBLE PRECISION ::
!     R(3) = RADIUS VECTOR (KM)
!     V(3) = VELOCITY VECTOR (KM/S)
! OUTPUT IS A DOUBLE PRECISION ARRAY WITH A LENGTH OF 6 ::
!     SV2OE(1) = a = SEMI-MAJOR AXIS (KM)
!     SV2OE(2) = e = ORBIT ECCENTRICITY
!     SV2OE(3) = i = ORBIT ECCENTRICITY (RAD)
!     SV2OE(4) = RA= LONGITUDE OF THE ASCENDING NODE (RAD)
```

```
!     SV2OE(5) = w = ARGUMENT OF PERIAPSIS (RAD)
!     SV2OE(6) = nu= TRUE ANOMALY (RAD)
!
! THE SUNS GRAVITATIONAL PARAMETER IS DEFINED AS A CONSTANT.  FOR
! OTHER BODIES THIS CAN BE CHANGED.
FUNCTION SV2OE(R,V)
IMPLICIT NONE

DOUBLE PRECISION, INTENT(IN) :: R(3),V(3)
DOUBLE PRECISION :: SV2OE(6),RMAG,VMAG,H, HVEC(3),EVEC(3),NVEC(3), &
    a,e,i,N, RA, w, NU, ENERGY

RMAG=NORM(R); VMAG=NORM(V)

HVEC=CROSS_PRODUCT(R,V); H=NORM(HVEC)
NVEC=CROSS_PRODUCT((/0.d0,0.d0,1.d0/),HVEC)
N=NORM(NVEC)
EVEC=((VMAG*VMAG-MU/RMAG)*R-DOT_PRODUCT(R,V)*V)/MU
e=NORM(EVEC)
ENERGY=VMAG*VMAG/2.d0-MU/RMAG
a=-MU/(2*ENERGY)

IF (e==1.d0) a=2.d8

i=ACOS(HVEC(3)/h)
RA=ACOS(NVEC(1)/N)

IF (NVEC(2)<0.d0) RA=2*PI-RA

w=ACOS(DOT_PRODUCT(NVEC,EVEC)/(N*e))

IF(EVEC(3)<0.d0) w=2*PI-w

NU=ACOS(DOT_PRODUCT(EVEC,R)/(e*RMAG))

IF (DOT_PRODUCT(R,V)<0.d0) NU=2*PI-NU

SV2OE(1)=a
SV2OE(2)=e
SV2OE(3)=i
SV2OE(4)=RA
SV2OE(5)=w
SV2OE(6)=NU

END FUNCTION SV2OE
```

APPENDIX B. FORTRAN CODE FOR THE HYBRID GNLP ALGORITHM

This appendix contains the code used for genetic algorithm and the wrappers for the three NLP solvers. The code is provided as is with no guarantees. If any of the codes or derivative works from codes in these appendices are used in another project the author simply asks that this thesis be referenced in any published works.

All of the included functions and subroutines have been tested to work with the GCC GFortran and Intel Fortran compilers. When used with the PGI or other Fortran compilers the intrinsic ISNAN function will likely cause problems.

```
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE IS THE PROGRAM THAT CONTROLS THE FLOW OF THE GENETIC
!  ALGORITHM.  THIS IS THE SUBROUTINE THAT SHOULD BE CALL BE THE
!  CALLED TO RUN THE GENETIC ALGORITHM.
!
!  INPUTS:
!    IPRINT        =   A VALUE GREATER THAN 0 PRINT INFORMATION FOR
!                      EACH GENERATION AS IT IS FINISH, INTEGER
!    N_POP         =   SIZE OF THE POPULATION, INTEGER
!    N_GEN         =   MAXIMUM NUMBER OF GENERATIONS TO RUN, INTEGER
!    N_INT         =   NUMBER OF INTEGER VARIABLES, INTEGER
!    N_DOUBLE      =   NUMBER OF REAL VALUED VARIABLES, INTEGER
!    N1            =   FIRST DIMENSION OF THE INPUT ARRAY, INTEGER
!    N2            =   SECOND DIMENSION OF THE INPUT ARRAY, INTEGER
!    ITER_MAX_NLP  =   MAXIMUM NUMBER OF ITERATIONS ALLOWED FOR THE
!                      SPECIFIED NLP SOLVER, INTEGER
!    N_CON         =   NUMBER OF CONSTRAINTS, CAN ONLY BE USED WITH
!                      COBYLA AND CONMIN, INTEGER
!    INTEGER_UPPER =   UPPER BOUNDS FOR INTEGER VARIABLES,
!                      INTEGER(N_INT)
!    INTEGER_LOWER =   LOWER BOUNDS FOR INTEGER VARIABLES,
```

```
!                       INTEGER(N_INT)
!    P_CROSS        =   PROBABILITY THAT A CROSSOVER WILL OCCUR, VALUES
!                       SHOULD TYPICALLY BE AROUND 0.9,
!                       DOUBLE PRECISION
!    P_REP          =   PROBABILITY THAT REPRODUCTIONS WILL OCCUR,
!                       VALUES SHOULD BE AROUND 0.1, DOUBLE PRECISION
!    P_MUT          =   PROBABILITY THAT A MUTATION WILL OCCUR, VALUES
!                       SHOULD BE TYPICALLY BE KEPT LOWER THAN 0.1,
!                       DOUBLE PRECISION
!    DOUBLE_UPPER   =   REAL VALUED VARIABLES UPPER BOUNDS,
!                       DOUBLE PRECISION(N_DOUBLE)
!    DOUBLE_LOWER   =   REAL VALUED VARIABLES LOWER BOUNDS,
!                       DOUBLE PRECISION(N_DOUBLE)
!    INPUT_ARRAY    =   INPUT ARRAY TO BE USED FOR ADDITIONAL INPUTS
!                       THAT THE COST FUNCTION MAY NEED,
!                       DOUBLE PRECISION (N1,N2)
!    CROSS_TYPE     =   TYPE OF CROSSOVER TO BE USED, OPTIONS ARE:
!                       UNIFORM, SINGLE_POINT, DOUBLE_POINT,
!                       ARITHMETIC,  AND HEURISTION,
!                       CHARACTER WITH A LENGTH OF 30
!    MUT_TYPE       =   TYPE OF MUTATION TO BE USED, OPTIONS ARE:
!                       UNIFORM, SLIDING, AND BOUNDARY,
!                       CHARACTER WITH A LENGTH OF 30
!    SEL_TYP        =   SELECTION TYPE TO BE USE, OPTIONS ARE:
!                       ROULETTE AND TOURNAMENT,
!                       CHARACTER WITH A LENGTH OF 30
!    OPT_TYPE       =   OPTIMIZATION TYPE TO BE USED, OPTIONS ARE:
!                       GEN, HYB_COBYLA, HYB_CONMIN, HYB_UNCMIN,
!                       CHARACTER WITH A LENGTH OF 30
!    SEED           =   SEED VALUE FOR THE RANDOM NUMBER GENERATOR,
!                       SHOULD BE STARTED WITH A NEGATIVE INTEGER
!                       VALUE, INTEGER
!
!  OUTPUTS:
!    FITNESS_MIN    =   ARRAY OF MINIMUM FITNESS VALUES FOR EACH
!                       GENERATION, DOUBLE PRECISION(N_GEN)
!    FITNESS_AVG    =   ARRAY OF THE AVERAGE FITNESS VALUES FOR EACH
!                       GENERATION, DOUBLE PRECISION(N_GEN)
!    INTEGER_MIN    =   INTEGER CHROMOSOME CORRESPONDING TO THE MINIMUM
!                       SOLUTION FOR EACH GENERATION,
!                       INTEGER(N_GEN, N_INT)
!    DOUBLE_MIN     =   REAL VALUES CHROMOSOME CORRESPONDING TO THE
!                       MINIMUM SOLUTION FOR EACH GENERATION,
!                       DOUBLE PRECISION(N_GEN,N_DOUBLE)
!
!*****************************************************************!
SUBROUTINE GENETIC_DRIVER(IPRINT, N_POP, N_GEN, N_INT, N_DOUBLE, N1, &
```

```
    N2, ITER_MAX_NLP, N_CON, INTEGER_UPPER, INTEGER_LOWER, P_CROSS, &
    P_REP,P_MUT, DOUBLE_UPPER, DOUBLE_LOWER, INPUT_ARRAY, CROSS_TYPE,&
    MUT_TYPE, SEL_TYPE, OPT_TYPE, SEED, FITNESS_MIN, FITNESS_AVG, &
    INTEGER_MIN, DOUBLE_MIN)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N_POP, N_GEN, N_INT, N_DOUBLE, N1, N2, N_CON, &
    INTEGER_UPPER(N_INT), INTEGER_LOWER(N_INT), IPRINT

INTEGER, INTENT(INOUT) :: SEED, ITER_MAX_NLP

DOUBLE PRECISION, INTENT(IN) :: P_CROSS, P_REP,P_MUT

DOUBLE PRECISION, INTENT(INOUT) :: DOUBLE_LOWER(N_DOUBLE), &
    DOUBLE_UPPER(N_DOUBLE), INPUT_ARRAY(N1,N2)

CHARACTER(LEN=30), INTENT(IN) :: CROSS_TYPE, MUT_TYPE, SEL_TYPE, &
    OPT_TYPE

DOUBLE PRECISION, INTENT(INOUT) :: FITNESS_MIN(N_GEN), &
    FITNESS_AVG(N_GEN), DOUBLE_MIN(N_GEN,N_DOUBLE)

INTEGER, INTENT(INOUT) :: INTEGER_MIN(N_GEN,N_INT)

DOUBLE PRECISION :: POP_DOUBLE(N_POP,N_DOUBLE), &
    POP_NEW_DOUBLE(N_POP,N_DOUBLE), FITNESS_INDV, FITNESS(N_POP), &
    FITNESS_NEW(N_POP), CHROM_DOUBLE(N_DOUBLE), TIME, &
    FITNESS_INDV_NLP, RAN, G_CON(N_CON)

INTEGER :: POP_INT(N_POP,N_INT), POP_NEW_INT(N_POP,N_INT), NAN_COUNT,&
    TEST, CHROM_INT(N_INT), P, Q, I, MIN_LOC, COUNT1, COUNT2, RATE, &
    ncon, count

DOUBLE PRECISION :: AVG, X0(N_DOUBLE), X(N_DOUBLE)

INTEGER :: NGEN_CONVERGE, INFO, NACMX1, N1_C, N2_C, N3_C, N4_C, N5_C

NCON=N_CON

CALL SYSTEM_CLOCK(COUNT1,RATE)

NAN_COUNT=0
NGEN_CONVERGE=50

!GENERATE INITIAL POPULATION AND FIND THEIR FITNESS VALUES
CALL POPULATION_GENERATOR(N_POP, N_DOUBLE, N_INT, POP_DOUBLE, &
    POP_INT, SEED, INTEGER_UPPER, INTEGER_LOWER, DOUBLE_UPPER, &
```

```
    DOUBLE_LOWER)

!$OMP PARALLEL DO private(CHROM_INT, CHROM_DOUBLE, FITNESS_INDV, X0)&
!$OMP& PRIVATE(X, G_CON, INPUT_ARRAY)

DO P=1,N_POP,1
    CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)
    CHROM_INT=POP_INT(P,1:N_INT)
    CALL COST(N_DOUBLE, N_INT, N1, N2, CHROM_DOUBLE, CHROM_INT, &
              FITNESS_INDV, INPUT_ARRAY, G_CON, NCON)
    ! THE INITIAL POPULATION SHOULD BE CHECKED TO ENSURE THAT THE COST
    ! FUNCTION DOESN'T GIVE ANY NANS OR VALUES THAT ARE TOO LARGER IF
    ! IT DOES A NEW CHROMOSOME IS RANDOMLY GENERATED UNTIL YOU FIND
    ! ONE THAT DOESN'T GIVE A NAN/VERY LARGE VALUES.
    IF (FITNESS_INDV .GE. 1.D24 .OR. ISNAN(FITNESS_INDV)) THEN
        COUNT=0
        DO WHILE(FITNESS_INDV .GE. 1.D24 .AND. COUNT.LT.50000)
            COUNT=COUNT+1
            CALL CHROMOSOME_GENERATOR(SEED, N_INT, N_DOUBLE, &
                CHROM_INT, CHROM_DOUBLE, INTEGER_UPPER, &
                INTEGER_LOWER, DOUBLE_LOWER, DOUBLE_UPPER)

            CALL COST(N_DOUBLE, N_INT, N1, N2, CHROM_DOUBLE, &
                CHROM_INT, FITNESS_INDV, INPUT_ARRAY, G_CON, NCON)
        END DO
        IF (COUNT.EQ.50000) THEN
            WRITE(*,*) "UNABLE TO FIND CHROMOSOME"
            WRITE(*,*) "THE PROBLEM MAY NOT BE WELL POSED"
        END IF
        POP_DOUBLE(P,1:N_DOUBLE)=CHROM_DOUBLE
        POP_INT(P,1:N_INT)=CHROM_INT
    END IF

    FITNESS(P)=FITNESS_INDV

END DO

!$OMP END PARALLEL DO
MIN_LOC=MINLOC(FITNESS,1)
FITNESS_MIN(1)=FITNESS(MIN_LOC)
INTEGER_MIN(1,1:N_INT)=POP_INT(MIN_LOC,1:N_INT)
DOUBLE_MIN(1,1:N_DOUBLE)=POP_DOUBLE(MIN_LOC,1:N_DOUBLE)
FITNESS_AVG(1)=SUM(FITNESS)/DBLE(N_POP)


CALL SYSTEM_CLOCK(COUNT2)
```

```
TIME=DBLE(COUNT2-COUNT1)/DBLE(RATE)
IF (IPRINT.GT.0) THEN
WRITE(*,*) ' GENERATION',' MINIMUM COST', &
    '              AVERAGE COST', '            NAN-COUNT     ',  &
    'RUN TIME'
WRITE(*,*) 1, FITNESS_MIN(1), FITNESS_AVG(1), NAN_COUNT, TIME
END IF

DO Q=2,N_GEN,1
    CALL GENETIC_OPERATIONS(FITNESS,FITNESS_NEW, POP_INT,&
                POP_NEW_INT, POP_DOUBLE, POP_NEW_DOUBLE, P_REP, &
                P_CROSS, P_MUT, N_POP, CROSS_TYPE, MUT_TYPE, &
                SEL_TYPE, N_INT, N_DOUBLE, SEED, INTEGER_UPPER, &
                INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER)

FITNESS=FITNESS_NEW
    ! IF THE CONMIN SOVLER IS BEING USES, EVEN THE TOP TWO SOLUTIONS
    ! SHOULD BE CHECKED TO ENSURE THAT CONSTRAINTS ARE MINIMIZED AS
    ! MUCH AS POSSIBLE
    IF(TRIM(OPT_TYPE).EQ."HYB_CONMIN") THEN
        FITNESS(1)=1.D8
        FITNESS(2)=1.D8
    END IF

    !$OMP PARALLEL DO PRIVATE(P, CHROM_INT, CHROM_DOUBLE)&
    !$OMP& PRIVATE(FITNESS_INDV, XO, X, FITNESS_INDV_NLP, G_CON)&
    !$OMP& PRIVATE(INPUT_ARRAY)
    DO P=1, N_POP,1

    !THE COST FUNCTION DOES NOT NEED TO BE CALCULATED FOR
        !INDIVIDUALS THAT WENT THROUGH REPRODUCTION, BUT NOT MUTATION.
        !IF AN INDIVIDUAL IN THE POPULATION IS FROM A  CROSSOVER  OR
        !MUTATION THE FITNESS VALUE IS SET TO 1.D8 AND A NEW FITNESS
        !VALUE SHOULD BE COMPUTED.  THIS ENSURE THAT NO UNNECESSARY
        !COST FUNCTION EVALUATIONS ARE PERFORMED.  THIS IS ALSO THE
        !PORTION  OF  THE  CODE  THAT SHOULD BE EXECUTED IN PARALLEL.
        !ALL OTHER PARTS  OF THE GENETIC  ALGORITHM TYPICALLY  REQUIRE
        !VERY LITTLE TIME TO EXECUTE COMPARED TO THE EVALUATION OF THE
        !FITNESS FUNCTION.
IF (ABS(FITNESS(P)-1.D8).LT.1.D0)THEN
            !MAKE SURE THAT THE NEW SOLUTION REGION IS FEASIBLE.  THIS
            !SECTION ALSO PERFORMS ALL THE NECESSARY CALCULATIONS FOR
            !THE PURE GENETIC ALGORITHM.   IF  ONE  OF  THE  HYBRID
            !METHODS IS UTILIZED FURTHER CALCULATIONS WILL THEN BE
            !PERFORMED.

            CHROM_INT=POP_NEW_INT(P,1:N_INT)
```

```
CHROM_DOUBLE=POP_NEW_DOUBLE(P,1:N_DOUBLE)
            CALL COST(N_DOUBLE, N_INT, N1, N2, CHROM_DOUBLE, &
                CHROM_INT, FITNESS_INDV, INPUT_ARRAY, G_CON, NCON)

            !NOW TEST TO MAKE SURE THE SOLUTION REGION HAS IS FEASIBLE
            !SOLUTION AT THE POINT IN THE POPULATION.  IF NOT THE
            !ORIGINAL SOLUTION FROM THE PRECIOUS POPULATION MEMBER P
            !WILL BE USED INSTEAD.
            IF (FITNESS_INDV.GE.1.D24 .OR. ISNAN(FITNESS_INDV)) THEN
                CHROM_INT=POP_INT(P,1:N_INT)
                CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)

                CALL COST(N_DOUBLE, N_INT, N1, N2, CHROM_DOUBLE, &
                    CHROM_INT, FITNESS_INDV, INPUT_ARRAY, G_CON, NCON)
                POP_NEW_INT(P,1:N_INT)=CHROM_INT
                POP_NEW_DOUBLE(P,1:N_INT)=CHROM_DOUBLE
            END IF

            IF(TRIM(OPT_TYPE).EQ."GEN") THEN
                !DO NOTHING, THE PURE GENETIC OPERATIONS HAVE ALREADY
                !BEEN PERFORMED
                X=CHROM_DOUBLE

            ELSE IF(TRIM(OPT_TYPE).EQ."HYB_UNCMIN") THEN
                !HYBRID ALGORITHM THAT USES AN UNCONSTRAINED
                !MINIMIZATION NLP SOVLER TO ITERATE ON THE REAL VALUED
                !CHROMOSOME

                X0=CHROM_DOUBLE
                CALL UNCMIN_WRAPPER(N_DOUBLE, N_INT, N1, N2, &
                    ITER_MAX_NLP, X0, X, CHROM_INT, FITNESS_INDV_NLP,&
                    INPUT_ARRAY, G_CON, NCON)
                !MAKE SURE THAT  THE  NEW  SOLUTION  IS FEASIBLE.
                !THIS SECTION CHECK  IF  THE  UNCMIN  CONVERGED  ON
                !A  SOLUTION (HOPEFULLY OPTIMAL, BUT  IT MAY  NOT  BE,
                !DON'T  WORRY  THE IDEA IS THE GENETIC  ALGORITHM
                !WILL  EVENTUALLY  CONVERGE ON AND OPTIMAL SOLUTION,
                !NOT  INDIVIDUAL  RUNS OF  THE NLP  SOLVER).  IF IT
                !DIDN'T THE COST FUNCTION SHOULD BE DESIGNED TO OUTPUT
                !A VALUELARGER THAN 1.D8.  THE ALGORITHM  ALSO CHECKS
                !FOR  NANS,  BUT THE  USER  DESIGNED  COST  FUNCTION
                !SHOULD BE DESIGNED NOT TO ALLOW ANY NANS (THIS WILL
                !LIKELY BREAK NEARLY ALL NLP SOLVERS WITH EVEN 1 NAN).

                DO I=1,N_DOUBLE,1
                    IF(X(I) .GT. DOUBLE_UPPER(I) .OR. &
                        X(I).LT.DOUBLE_LOWER(I)) THEN
```

```
            FITNESS_INDV_NLP=1.D30
        END IF
    END DO

    IF(FITNESS_INDV_NLP .GE. 1.D24 .OR. &
        ISNAN(FITNESS_INDV_NLP))THEN
        !UNCMIN FAILED, SO THE SOLUTION FROM THE PREVIOUS
        !GENERATION WILL BE USED INSTEAD.
        CHROM_INT=POP_INT(P,1:N_INT)
        CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)
        POP_NEW_DOUBLE(P,1:N_DOUBLE)=CHROM_DOUBLE
        POP_NEW_INT(P,1:N_INT)=CHROM_INT
    ELSE

        !THE SOLUTION FOUND BY UNCMIN IS VALID, SO THE
        !NEW POPULATION  WITH  THE  DETERMINED SOLUTION
        !AND COST FUNCTION IS UPDATED.
        POP_NEW_DOUBLE(P,1:N_DOUBLE)=X
        FITNESS_INDV=FITNESS_INDV_NLP
    END IF
ELSE IF(TRIM(OPT_TYPE).EQ."HYB_CONMIN") THEN
    !HYBRID ALGORITHM THAT USES A CONMIN

    X0=CHROM_DOUBLE

    CALL CONMIN_WRAPPER(N_DOUBLE, N_INT, N1, N2, &
        ITER_MAX_NLP, X0, X, CHROM_INT, FITNESS_INDV_NLP,&
        INPUT_ARRAY, N_CON, DOUBLE_UPPER, DOUBLE_LOWER)

    IF(FITNESS_INDV_NLP .GE. 1.D24 .OR. &
        ISNAN(FITNESS_INDv_NLP))THEN
        !CONMIN FAILED, SO THE PREVIOUS GENERATION WILL
        !BE USED INSTEAD.
        CHROM_INT=POP_INT(P,1:N_INT)
        CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)
        POP_NEW_DOUBLE(P,1:N_DOUBLE)=CHROM_DOUBLE
        POP_NEW_INT(P,1:N_INT)=CHROM_INT
    ELSE

        !THE SOLUTION FOUND BY CONMIN IS VALID, SO THE NEW
        !POPULATION  WITH  THE  DETERMINED SOLUTION AND
        !COST FUNCTION IS UPDATED.
        POP_NEW_DOUBLE(P,1:N_DOUBLE)=X
        FITNESS_INDV=FITNESS_INDV_NLP
    END IF
ELSE IF(TRIM(OPT_TYPE).EQ."HYB_COBYLA") THEN
    !HYBRID ALGORITHM THAT USES A CONSTRAINED MINIMIZATION
```

```
                !WRITE(*,*) "CALLING CONMIN WRAPPER"

                X=CHROM_DOUBLE
                CALL COBYLA_DRIVER(N_DOUBLE, N_INT, N1, N2, X, &
                    CHROM_INT, FITNESS_INDV_NLP, INPUT_ARRAY, N_CON,&
                    ITER_MAX_NLP )

                DO I=1,N_DOUBLE,1
                    IF(X(I) .GT. DOUBLE_UPPER(I) .OR. &
                        X(I).LT.DOUBLE_LOWER(I)) THEN
                        FITNESS_INDV_NLP=1.D30
                    END IF
                END DO

                IF(FITNESS_INDV_NLP .GE. 1.D24 .OR. &
                    ISNAN(FITNESS_INDv_NLP))THEN
                    !COBYLA.
                    CHROM_INT=POP_INT(P,1:N_INT)
                    CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)
                    POP_NEW_DOUBLE(P,1:N_DOUBLE)=CHROM_DOUBLE
                    POP_NEW_INT(P,1:N_INT)=CHROM_INT
                ELSE

                    !THE SOLUTION FOUND BY CONMIN IS VALID, SO THE NEW
                    !POPULATION  WITH  THE  DETERMINED SOLUTION AND
                    !COST FUNCTION IS UPDATED.
                    POP_NEW_DOUBLE(P,1:N_DOUBLE)=X
                    FITNESS_INDV=FITNESS_INDV_NLP
                END IF
            END IF



FITNESS(P)=FITNESS_INDV
        END IF
END DO
    !$OMP END PARALLEL DO


POP_INT=POP_NEW_INT
POP_DOUBLE=POP_NEW_DOUBLE
MIN_LOC=MINLOC(FITNESS,1)
FITNESS_MIN(Q)=FITNESS(MIN_LOC)

    !CHECK TO MAKE SURE THE SOLUTION HASN'T STAGNATED FOR MORE THAN
    !25 GENERATIONS
    AVG=1.D0
```

```
    IF (Q>NGEN_CONVERGE) THEN
        AVG=ABS(SUM(FITNESS_MIN(Q-NGEN_CONVERGE:Q))-&
            SUM(FITNESS_MIN(Q-(NGEN_CONVERGE+1):Q-1)))
        IF (AVG.LE.1.D-1) THEN
            DO I=Q,N_GEN,1
               INTEGER_MIN(I,1:N_INT)=POP_INT(MIN_LOC,1:N_INT)
               DOUBLE_MIN(I,1:N_DOUBLE)=POP_DOUBLE(MIN_LOC,1:N_DOUBLE)
               FITNESS_MIN(I)=FITNESS(MIN_LOC)
            END DO
            EXIT
        END IF

    END IF

INTEGER_MIN(Q,1:N_INT)=POP_INT(MIN_LOC,1:N_INT)
DOUBLE_MIN(Q,1:N_DOUBLE)=POP_DOUBLE(MIN_LOC,1:N_DOUBLE)
FITNESS_AVG(Q)=SUM(FITNESS)/DBLE(N_POP)
    CALL SYSTEM_CLOCK(COUNT2)

    TIME=DBLE(COUNT2-COUNT1)/DBLE(RATE)
    IF (IPRINT.GT.0) THEN
    WRITE(*,*) Q, FITNESS_MIN(Q), FITNESS_AVG(Q), NAN_COUNT, TIME
    END IF
END DO

END SUBROUTINE GENETIC_DRIVER


!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE PERFORMS ALL THE GENETIC OPERATIONS ON THE INITIAL
!  POPULATION FOR EACH GENERATION
SUBROUTINE GENETIC_OPERATIONS(FITNESS,FITNESS_NEW, POP_INT,&
    POP_NEW_INT, POP_DOUBLE, POP_NEW_DOUBLE, P_REP, P_CROSS, P_MUT, &
    N_POP, CROSS_TYPE,  MUT_TYPE, SEL_TYPE, N_INT, N_DOUBLE, SEED, &
    INTEGER_UPPER, INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N_POP, N_INT, N_DOUBLE, POP_INT(N_POP,N_INT), &
    INTEGER_UPPER(N_INT), INTEGER_LOWER(N_INT)
INTEGER, INTENT(INOUT) :: SEED, POP_NEW_INT(N_POP,N_INT)
DOUBLE PRECISION, INTENT(IN) :: FITNESS(N_POP), P_REP, P_CROSS, &
    POP_DOUBLE(N_POP,N_DOUBLE), P_MUT, DOUBLE_UPPER(N_DOUBLE), &
    DOUBLE_LOWER(N_DOUBLE)
DOUBLE PRECISION, INTENT(INOUT) :: FITNESS_NEW(N_POP), &
    POP_NEW_DOUBLE(N_POP,N_DOUBLE)
```

```
CHARACTER(LEN=30), INTENT(IN) :: CROSS_TYPE, MUT_TYPE, SEL_TYPE
DOUBLE PRECISION :: RAN, CHROM_DOUBLE(N_DOUBLE)
INTEGER :: P, CHROM_INT(N_INT)

IF(TRIM(SEL_TYPE).EQ."ROULETTE")THEN
CALL ROULETTE(SEED, N_POP, N_INT, N_DOUBLE, FITNESS, FITNESS_NEW,&
        POP_INT, POP_NEW_INT, POP_DOUBLE, POP_NEW_DOUBLE, P_CROSS, &
        P_REP, CROSS_TYPE, INTEGER_UPPER, INTEGER_LOWER, &
        DOUBLE_UPPER, DOUBLE_LOWER)
ELSE IF(TRIM(SEL_TYPE).EQ."TOURNAMENT")THEN
CALL TOURNAMENT(SEED, N_POP, N_INT, N_DOUBLE, FITNESS, &
        FITNESS_NEW, POP_INT, POP_NEW_INT, POP_DOUBLE, &
        POP_NEW_DOUBLE, P_CROSS, P_REP, CROSS_TYPE, INTEGER_UPPER, &
        INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER)
ELSE
!WRITE(*,*) "INVALID SELECTION TYPE"
END IF

! AFTER NEW POPULATION HAS BEEN GENERATED THE MUTATIONS ARE DONE. THE
! MUTATION PROBABILITY SHOULD BE USED RELATIVELY LOW TO AVOID A
! COMPLETELY RANDOM SEARCH
DO P=3,N_POP,1
RAN=RANDOM(SEED)
IF (RAN.LE.P_MUT) THEN
CHROM_INT=POP_INT(P,1:N_INT)
CHROM_DOUBLE=POP_DOUBLE(P,1:N_DOUBLE)
CALL MUTATION(CHROM_INT, CHROM_DOUBLE, N_INT, N_DOUBLE, SEED, &
            MUT_TYPE, INTEGER_LOWER, INTEGER_UPPER, DOUBLE_LOWER, &
            DOUBLE_UPPER)
POP_NEW_INT(P,1:N_INT)=CHROM_INT
POP_NEW_DOUBLE(P,1:N_DOUBLE)=CHROM_DOUBLE
FITNESS_NEW(P)=1.D8
END IF
END DO

END SUBROUTINE GENETIC_OPERATIONS

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE PERFORMES THE ROULETTE SELECTION METHOD OPERATOR
!
SUBROUTINE ROULETTE(SEED, N_POP, N_INT, N_DOUBLE, FITNESS, &
    FITNESS_NEW, POP_INT, POP_NEW_INT, POP_DOUBLE, POP_NEW_DOUBLE, &
    P_CROSS, P_REP, CROSS_TYPE, INTEGER_UPPER, INTEGER_LOWER, &
    DOUBLE_UPPER, DOUBLE_LOWER)
IMPLICIT NONE
```

```
INTEGER, INTENT(IN) :: N_POP, N_INT, N_DOUBLE, POP_INT(N_POP, N_INT)

INTEGER, INTENT(INOUT) :: SEED, POP_NEW_INT(N_POP, N_INT)

DOUBLE PRECISION, INTENT(IN) :: P_CROSS, P_REP, FITNESS(N_POP), &
    POP_DOUBLE(N_POP, N_DOUBLE)

DOUBLE PRECISION, INTENT(INOUT) :: FITNESS_NEW(N_POP), &
    POP_NEW_DOUBLE(N_POP,N_DOUBLE)

CHARACTER(LEN=30), INTENT(IN) :: CROSS_TYPE

INTEGER, INTENT(IN) :: INTEGER_UPPER(N_INT), INTEGER_LOWER(N_INT)

DOUBLE PRECISION, INTENT(IN) :: DOUBLE_UPPER(N_DOUBLE), &
    DOUBLE_LOWER(N_DOUBLE)

DOUBLE PRECISION :: RAN, RAN1, RAN2, SUM, NORM_SWAP(N_POP), &
    FITNESS_SWAP(N_POP), NORMAL(N_POP), STD(N_POP), ADJ(N_POP), &
    SUM_ADJ, CHROM1_DOUBLE(N_DOUBLE), CHROM2_DOUBLE(N_DOUBLE), &
    POP_SWAP_DOUBLE(N_POP,N_DOUBLE), FIT1, FIT2
INTEGER :: TEST1, TEST2, ITER, CHROM1_INT(N_INT), CHROM2_INT(N_INT),&
    INDEX_SORTED(N_POP), I, POP_SWAP_INT(N_POP,N_INT), N, INDEX1, &
    INDEX2
! COMPUTE THE NORMALIZED DISTRIBUTION HERE
SUM_ADJ=0.D0
DO I=1,N_POP,1
    STD(I)=FITNESS(I)
    ADJ(I)=1.D0/(1.D0+STD(I))
    SUM_ADJ=SUM_ADJ+ADJ(I)
END DO

DO I=1,N_POP,1
    NORMAL(I)=ADJ(I)/SUM_ADJ
INDEX_SORTED(I)=I
END DO

CALL HEAP_SORT(NORMAL,INDEX_SORTED,N_POP)

! THE NORM VECTOR IS RETURNED FROM SMALLEST TO LARGEST, BUT NORM NEED
! TO BE SORTED FROMLARGEST TO SMALLEST, SO IT IS REVERSED HERE, ALONG
!  WITH THE POPULATION INTO POP_SWAP
N=N_POP
DO I=1,N_POP,1
  NORM_SWAP(I)=NORMAL(N)
  POP_SWAP_DOUBLE(I,1:N_DOUBLE)=POP_DOUBLE(INDEX_SORTED(N),1:N_DOUBLE)
```

```
  POP_SWAP_INT(I,1:N_INT)=POP_INT(INDEX_SORTED(N),1:N_INT)
  FITNESS_SWAP(I)=FITNESS(INDEX_SORTED(N))
  N=N-1
END DO
!WRITE(*,*) "SWAPPING DONE"
NORMAL=NORM_SWAP
POP_NEW_INT(1,1:N_INT)=POP_SWAP_INT(1,1:N_INT)
POP_NEW_INT(2,1:N_INT)=POP_SWAP_INT(2,1:N_INT)
POP_NEW_DOUBLE(1,1:N_DOUBLE)=POP_SWAP_DOUBLE(1,1:N_DOUBLE)
POP_NEW_DOUBLE(2,1:N_DOUBLE)=POP_SWAP_DOUBLE(2,1:N_DOUBLE)


FITNESS_NEW(1)=FITNESS_SWAP(1)
FITNESS_NEW(2)=FITNESS_SWAP(2)

!WRITE(*,*) "STARTING ROULETTE SELECTION"

DO I=3,N_POP,2
    !WRITE(*,*) "I=", I

!ROULETTE SELECTION IS PERFORMED HERE
RAN1=RANDOM(SEED)
RAN2=RANDOM(SEED)
SUM=0.DO
ITER=0
TEST1=0
TEST2=0
DO WHILE (TEST1.EQ.0 .OR. TEST2.EQ.0)
ITER=ITER+1
SUM=SUM+NORMAL(ITER)
IF (SUM.GT.RAN1 .AND. TEST1.EQ.0) THEN
INDEX1=ITER
TEST1=1
END IF
IF (SUM.GT.RAN2 .AND. TEST2.EQ.0) THEN
INDEX2=ITER
TEST2=1
END IF
    END DO

    !WRITE(*,*) " REPRODUCTION AND CROSSOVER STARTED"
!REPRODUCTION AND CROSSOVER CAN BE PERFORMED NOW THAT 2 PARENTS
    !HAVE BEEN CHOSEN
RAN=RANDOM(SEED)
IF (RAN>P_CROSS)THEN
POP_NEW_INT(I,1:N_INT)=POP_SWAP_INT(INDEX1,1:N_INT)
POP_NEW_INT(I+1,1:N_INT)=POP_SWAP_INT(INDEX2,1:N_INT)
```

```
POP_NEW_DOUBLE(I,1:N_DOUBLE)=&
            POP_SWAP_DOUBLE(INDEX1,1:N_DOUBLE)
POP_NEW_DOUBLE(I+1,1:N_DOUBLE)=&
            POP_SWAP_DOUBLE(INDEX2,1:N_DOUBLE)
        FITNESS_NEW(I)=FITNESS_SWAP(INDEX1)
        FITNESS_NEW(I+1)=FITNESS_SWAP(INDEX2)
ELSE
CHROM1_INT=POP_SWAP_INT(INDEX1,1:N_INT)
CHROM2_INT=POP_SWAP_INT(INDEX2,1:N_INT)
CHROM1_DOUBLE=POP_SWAP_DOUBLE(INDEX1,1:N_DOUBLE)
CHROM2_DOUBLE=POP_SWAP_DOUBLE(INDEX2,1:N_DOUBLE)
FIT1=NORMAL(INDEX1)
FIT2=NORMAL(INDEX2)

        CALL CROSSOVER(CHROM1_INT, CHROM2_INT, CHROM1_DOUBLE, &
            CHROM2_DOUBLE, N_INT, N_DOUBLE, SEED, CROSS_TYPE, &
            INTEGER_UPPER, INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER,&
            FIT1, FIT2)

POP_NEW_INT(I,1:N_INT)=CHROM1_INT
POP_NEW_INT(I+1,1:N_INT)=CHROM2_INT
POP_NEW_DOUBLE(I,1:N_DOUBLE)=CHROM1_DOUBLE
POP_NEW_DOUBLE(I+1,1:N_DOUBLE)=CHROM2_DOUBLE
FITNESS_NEW(I)=1.D8
FITNESS_NEW(I+1)=1.D8
END IF

END DO

END SUBROUTINE ROULETTE

!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
! THIS SUBROUTINE EXECUTES THE TOURNAMENT SELECTION OPERATOR
!
SUBROUTINE TOURNAMENT(SEED, N_POP, N_INT, N_DOUBLE, FITNESS, &
    FITNESS_NEW, POP_INT, POP_NEW_INT, POP_DOUBLE, POP_NEW_DOUBLE, &
    P_CROSS, P_REP, CROSS_TYPE,  INTEGER_UPPER, INTEGER_LOWER, &
    DOUBLE_UPPER, DOUBLE_LOWER)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N_POP, N_INT, N_DOUBLE, POP_INT(N_POP, N_INT),&
    INTEGER_UPPER(N_INT), INTEGER_LOWER(N_INT)

INTEGER, INTENT(INOUT) :: SEED, POP_NEW_INT(N_POP, N_INT)
```

```
DOUBLE PRECISION, INTENT(IN) :: P_CROSS, P_REP, FITNESS(N_POP), &
    POP_DOUBLE(N_POP, N_DOUBLE), DOUBLE_UPPER(N_DOUBLE), &
    DOUBLE_LOWER(N_DOUBLE)

DOUBLE PRECISION, INTENT(INOUT) :: FITNESS_NEW(N_POP), &
    POP_NEW_DOUBLE(N_POP,N_DOUBLE)

CHARACTER(LEN=30), INTENT(IN) :: CROSS_TYPE

DOUBLE PRECISION :: NORMAL(N_POP), STD(N_POP), ADJ(N_POP), SUM_ADJ, &
    RAN, POP_DOUBLE_POOL(N_POP,N_DOUBLE), FITNESS_POOL(N_POP)

INTEGER :: I, N, INDEX, ORDER1(N_POP), ORDER2(N_POP), &
    LOCATION(N_POP), LOCATION_SWAP(N_POP), POP_INT_POOL(N_POP,N_INT)



! COMPUTE THE NORMALIZED DISTRIBUTION HERE
SUM_ADJ=0.D0
DO I=1,N_POP,1
STD(I)=FITNESS(I)
ADJ(I)=1.D0/(1.D0+STD(I))
SUM_ADJ=SUM_ADJ+ADJ(I)
END DO

DO I=1,N_POP,1
    NORMAL(I)=ADJ(I)/SUM_ADJ
END DO

! FIRST RANDOM ORDER LOCATION
DO I=1,N_POP,1
LOCATION(I)=I
END DO

LOCATION_SWAP=LOCATION
N=N_POP

DO I=1,N_POP,1
RAN=RANDOM(SEED)
INDEX=FLOOR(RAN*DBLE(N)+1.D0)
ORDER1(I)=LOCATION(INDEX)
LOCATION_SWAP(1:INDEX-1)=LOCATION(1:INDEX-1)
LOCATION_SWAP(INDEX:N-1)=LOCATION(INDEX+1:N)
LOCATION=LOCATION_SWAP
N=N-1
END DO
```

```
! SECOND RANDOM ORDER LOCATION
DO I=1,N_POP,1
LOCATION(I)=I
END DO

LOCATION_SWAP=LOCATION
N=N_POP

DO I=1,N_POP,1
RAN=RANDOM(SEED)
INDEX=FLOOR(RAN*DBLE(N)+1.DO)
ORDER2(I)=LOCATION(INDEX)
LOCATION_SWAP(1:INDEX-1)=LOCATION(1:INDEX-1)
LOCATION_SWAP(INDEX:N-1)=LOCATION(INDEX+1:N)
LOCATION=LOCATION_SWAP
N=N-1
END DO

!PERFORM TOURNAMENT PRIOR TO ROULETTE SELECTION
DO I=1,N_POP,2
IF(NORMAL(ORDER1(I)).GT.NORMAL(ORDER1(I+1)))THEN
POP_INT_POOL(I,1:N_INT)=POP_INT(ORDER1(I),1:N_INT)
POP_DOUBLE_POOL(I,1:N_DOUBLE)=&
                POP_DOUBLE(ORDER1(I),1:N_DOUBLE)
FITNESS_POOL(I)=FITNESS(ORDER1(I))
ELSE
POP_INT_POOL(I,1:N_INT)=POP_INT(ORDER1(I+1),1:N_INT)
POP_DOUBLE_POOL(I,1:N_DOUBLE)=&
                POP_DOUBLE(ORDER1(I+1),1:N_DOUBLE)
FITNESS_POOL(I)=FITNESS(ORDER1(I+1))
END IF

IF(NORMAL(ORDER2(I)).GT.NORMAL(ORDER2(I+1)))THEN
POP_INT_POOL(I+1,1:N_INT)=POP_INT(ORDER2(I),1:N_INT)
POP_DOUBLE_POOL(I+1,1:N_DOUBLE)=&
                POP_DOUBLE(ORDER2(I),1:N_DOUBLE)
FITNESS_POOL(I+1)=FITNESS(ORDER2(I))
ELSE
POP_INT_POOL(I+1,1:N_INT)=POP_INT(ORDER2(I+1),1:N_INT)
POP_DOUBLE_POOL(I+1,1:N_DOUBLE)=&
                POP_DOUBLE(ORDER2(I+1),1:N_DOUBLE)
FITNESS_POOL(I+1)=FITNESS(ORDER2(I+1))
END IF
END DO

CALL ROULETTE(SEED, N_POP, N_INT, N_DOUBLE, FITNESS_POOL, &
    FITNESS_NEW, POP_INT_POOL, POP_NEW_INT, POP_DOUBLE_POOL, &
```

```
    POP_NEW_DOUBLE, P_CROSS, P_REP, CROSS_TYPE, INTEGER_UPPER, &
    INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER)

END SUBROUTINE TOURNAMENT

!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
! THIS SUBROUTINE EXECUTES THE CROSSOVER OPERATORS
!
SUBROUTINE CROSSOVER(CHROM1_INT, CHROM2_INT, CHROM1_DOUBLE, &
    CHROM2_DOUBLE, N_INT, N_DOUBLE, SEED, CROSS_TYPE, INTEGER_UPPER,&
    INTEGER_LOWER, DOUBLE_UPPER, DOUBLE_LOWER, FIT1, FIT2)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N_INT, N_DOUBLE, INTEGER_UPPER(N_INT), &
    INTEGER_LOWER(N_INT)

INTEGER, INTENT(INOUT) :: SEED, CHROM1_INT(N_INT), CHROM2_INT(N_INT)

CHARACTER(LEN=30), INTENT(IN) :: CROSS_TYPE

DOUBLE PRECISION, INTENT(INOUT) :: CHROM1_DOUBLE(N_DOUBLE), &
    CHROM2_DOUBLE(N_DOUBLE)

DOUBLE PRECISION, INTENT(IN) :: DOUBLE_UPPER(N_DOUBLE), &
    DOUBLE_LOWER(N_DOUBLE), FIT1, FIT2

INTEGER :: I_DBLE, I, N, CHROM1_INT_NEW(N_INT), CHROM2_INT_NEW(N_INT),&
    LOC1, LOC2, LOC_SWAP, CHROM_INT_BEST(N_INT), &
    CHROM_INT_WORST(N_INT), TEST1, TEST2, COUNT

DOUBLE PRECISION :: RAN, CHROM1_DOUBLE_NEW(N_DOUBLE), &
    CHROM2_DOUBLE_NEW(N_DOUBLE), CHROM_DOUBLE_BEST(N_DOUBLE), &
    CHROM_DOUBLE_WORST(N_DOUBLE)



N=N_INT+N_DOUBLE
I_DBLE=0

IF(TRIM(CROSS_TYPE).EQ."UNIFORM")THEN
DO I=1,N_INT,1
IF(CHROM1_INT(I).EQ.CHROM2_INT(I))THEN
            CHROM1_INT_NEW(I)=CHROM1_INT(I)
CHROM2_INT_NEW(I)=CHROM2_INT(I)
        ELSE
```

```
RAN=RANDOM(SEED)
IF(RAN.LE.0.5D0)THEN
CHROM1_INT_NEW(I)=CHROM1_INT(I)
CHROM2_INT_NEW(I)=CHROM2_INT(I)
ELSE
CHROM1_INT_NEW(I)=CHROM2_INT(I)
CHROM2_INT_NEW(I)=CHROM1_INT(I)
END IF
END IF
    END DO

    DO I=1,N_DOUBLE,1
RAN=RANDOM(SEED)
        IF(RAN.LE.0.5D0)THEN
CHROM1_DOUBLE_NEW(I)=CHROM1_DOUBLE(I)
CHROM2_DOUBLE_NEW(I)=CHROM2_DOUBLE(I)
ELSE
CHROM1_DOUBLE_NEW(I)=CHROM2_DOUBLE(I)
CHROM2_DOUBLE_NEW(I)=CHROM1_DOUBLE(I)
END IF
END DO

CHROM1_INT=CHROM1_INT_NEW
CHROM2_INT=CHROM2_INT_NEW
CHROM1_DOUBLE=CHROM1_DOUBLE_NEW
CHROM2_DOUBLE=CHROM2_DOUBLE_NEW

ELSE IF(TRIM(CROSS_TYPE).EQ."SINGLE_POINT")THEN

LOC1=RANDOM_INTEGER(SEED,N,0)

    IF (LOC1.EQ.0) LOC1=1

DO I=1,N,1
IF(I.LE.LOC1)THEN

IF(I.LE.N_INT)THEN
CHROM1_INT_NEW(I)=CHROM1_INT(I)
CHROM2_INT_NEW(I)=CHROM2_INT(I)
ELSE
CHROM1_DOUBLE_NEW(I-N_INT)=CHROM1_DOUBLE(I-N_INT)
CHROM2_DOUBLE_NEW(I-N_INT)=CHROM2_DOUBLE(I-N_INT)
END IF

ELSE

IF(I.LE.N_INT)THEN
```

```
CHROM1_INT_NEW(I)=CHROM2_INT(I)
CHROM2_INT_NEW(I)=CHROM1_INT(I)
ELSE
CHROM1_DOUBLE_NEW(I-N_INT)=CHROM2_DOUBLE(I-N_INT)
CHROM2_DOUBLE_NEW(I-N_INT)=CHROM1_DOUBLE(I-N_INT)
END IF

END IF
END DO

CHROM1_INT=CHROM1_INT_NEW
CHROM2_INT=CHROM2_INT_NEW
CHROM1_DOUBLE=CHROM1_DOUBLE_NEW
CHROM2_DOUBLE=CHROM2_DOUBLE_NEW


ELSE IF(TRIM(CROSS_TYPE).EQ."DOUBLE_POINT")THEN

LOC1=RANDOM_INTEGER(SEED,N,0)
    IF (LOC1.EQ.0) LOC1=1
LOC2=RANDOM_INTEGER(SEED,N,0)
    IF (LOC2.EQ.0) LOC2=1
!MAKE SURE THE TWO LOCATIONS AREN'T THE SAME

DO  WHILE(LOC1.EQ.LOC2)
LOC2=RANDOM_INTEGER(SEED,N,0)
        IF (LOC2.EQ.0) LOC2=1
END DO

IF(LOC1.GT.LOC2)THEN
LOC_SWAP=LOC2
LOC2=LOC1
LOC1=LOC_SWAP
END IF

DO I=1,N,1
IF(I.LE.LOC1)THEN

IF(I.LE.N_INT)THEN
CHROM1_INT_NEW(I)=CHROM1_INT(I)
CHROM2_INT_NEW(I)=CHROM2_INT(I)
ELSE
CHROM1_DOUBLE_NEW(I-N_INT)=CHROM1_DOUBLE(I-N_INT)
CHROM2_DOUBLE_NEW(I-N_INT)=CHROM2_DOUBLE(I-N_INT)
END IF

ELSE IF(I.GT.LOC2)THEN
```

```
IF(I.LE.N_INT)THEN
CHROM1_INT_NEW(I)=CHROM1_INT(I)
CHROM2_INT_NEW(I)=CHROM2_INT(I)
ELSE
CHROM1_DOUBLE_NEW(I-N_INT)=CHROM1_DOUBLE(I-N_INT)
CHROM2_DOUBLE_NEW(I-N_INT)=CHROM2_DOUBLE(I-N_INT)
END IF

ELSE

IF(I.LE.N_INT)THEN
CHROM1_INT_NEW(I)=CHROM2_INT(I)
CHROM2_INT_NEW(I)=CHROM1_INT(I)
ELSE
CHROM1_DOUBLE_NEW(I-N_INT)=CHROM2_DOUBLE(I-N_INT)
CHROM2_DOUBLE_NEW(I-N_INT)=CHROM1_DOUBLE(I-N_INT)
END IF

END IF
END DO

CHROM1_INT=CHROM1_INT_NEW
CHROM2_INT=CHROM2_INT_NEW
CHROM1_DOUBLE=CHROM1_DOUBLE_NEW
CHROM2_DOUBLE=CHROM2_DOUBLE_NEW

ELSE IF(TRIM(CROSS_TYPE).EQ."ARITHMETIC")THEN

RAN=RANDOM(SEED)
CHROM1_INT_NEW=INT(RAN*DBLE(CHROM1_INT)+&
        (1.DO-RAN)*DBLE(CHROM2_INT))
CHROM2_INT_NEW=INT((1.DO-RAN)*DBLE(CHROM1_INT)+&
        RAN*DBLE(CHROM2_INT))
CHROM1_DOUBLE_NEW=RAN*CHROM1_DOUBLE+(1.DO-RAN)*CHROM2_DOUBLE
CHROM2_DOUBLE_NEW=(1.DO-RAN)*CHROM1_DOUBLE+RAN*CHROM2_DOUBLE

!TEST TO MAKE SURE THE RESULTING CHROMOSOMES AREN'T OUTSIDE THE
    !ALLOWED BOUNDS
DO I=1,N_INT,1
IF(CHROM1_INT_NEW(I).LT.INTEGER_LOWER(I)) THEN
CHROM1_INT_NEW(I)=INTEGER_LOWER(I)
ELSE IF(CHROM1_INT_NEW(I).GT.INTEGER_UPPER(I))THEN
CHROM1_INT_NEW(I)=INTEGER_UPPER(I)
END IF

IF(CHROM2_INT_NEW(I).LT.INTEGER_LOWER(I)) THEN
```

```
CHROM2_INT_NEW(I)=INTEGER_LOWER(I)
ELSE IF(CHROM2_INT_NEW(I).GT.INTEGER_UPPER(I))THEN
CHROM2_INT_NEW(I)=INTEGER_UPPER(I)
END IF
END DO

CHROM1_INT=CHROM1_INT_NEW
CHROM2_INT=CHROM2_INT_NEW
CHROM1_DOUBLE=CHROM1_DOUBLE_NEW
CHROM2_DOUBLE=CHROM2_DOUBLE_NEW

ELSE IF(TRIM(CROSS_TYPE).EQ."HEURISTIC")THEN

IF(FIT1.LT.FIT2)THEN
CHROM_INT_BEST=CHROM1_INT
CHROM_INT_WORST=CHROM2_INT
CHROM_DOUBLE_BEST=CHROM1_DOUBLE
CHROM_DOUBLE_WORST=CHROM2_DOUBLE
ELSE
CHROM_INT_BEST=CHROM2_INT
CHROM_INT_WORST=CHROM1_INT
CHROM_DOUBLE_BEST=CHROM2_DOUBLE
CHROM_DOUBLE_WORST=CHROM1_DOUBLE
END IF

TEST1=0
COUNT=1

DO WHILE (TEST1.EQ.0 .AND. COUNT.LT.50)
RAN=RANDOM(SEED)

CHROM1_INT_NEW=CHROM_INT_BEST+&
            INT(RAN*DBLE(CHROM_INT_BEST-CHROM_INT_WORST))
CHROM1_DOUBLE_NEW=CHROM_DOUBLE_BEST+&
            RAN*(CHROM_DOUBLE_BEST-CHROM_DOUBLE_WORST)

CHROM2_INT_NEW=CHROM_INT_BEST
CHROM2_DOUBLE_NEW=CHROM_DOUBLE_BEST

TEST2=0

DO I=I,N_INT,1

IF(CHROM1_INT(I).LT.INTEGER_LOWER(I))THEN
TEST2=1
ELSE IF(CHROM1_INT(I).GT.INTEGER_UPPER(I))THEN
TEST2=1
```

```
END IF

IF(CHROM2_INT(I).LT.INTEGER_LOWER(I))THEN
TEST2=1
ELSE IF(CHROM2_INT(I).GT.INTEGER_UPPER(I))THEN
TEST2=1
END IF

END DO

DO I=1,N_DOUBLE,1

IF(CHROM1_DOUBLE(I).LT.DOUBLE_LOWER(I))THEN
TEST2=1
ELSE IF(CHROM1_DOUBLE(I).GT.DOUBLE_UPPER(I))THEN
TEST2=1
END IF

IF(CHROM2_DOUBLE(I).LT.DOUBLE_LOWER(I))THEN
TEST2=1
ELSE IF(CHROM2_DOUBLE(I).GT.DOUBLE_UPPER(I))THEN
TEST2=1
END IF

END DO

IF(TEST2.EQ.0)THEN
TEST1=1
END IF

COUNT=COUNT+1

END DO

IF(COUNT.GE.50) THEN
CHROM1_INT_NEW=CHROM_INT_WORST
CHROM1_DOUBLE_NEW=CHROM_DOUBLE_WORST
END IF

CHROM1_INT=CHROM1_INT_NEW
CHROM2_INT=CHROM2_INT_NEW
CHROM1_DOUBLE=CHROM1_DOUBLE_NEW
CHROM2_DOUBLE=CHROM2_DOUBLE_NEW

ELSE
WRITE(*,*) "INVALID CROSSOVER TYPE: NO CROSSOVER PERFORMED"
END IF
```

```
END SUBROUTINE CROSSOVER

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE EXECUTES THE MUTATION OPERATOR
SUBROUTINE MUTATION(CHROM_INT, CHROM_DOUBLE, N_INT, N_DOUBLE, SEED, &
  MUT_TYPE, INTEGER_LOWER, INTEGER_UPPER, DOUBLE_LOWER, DOUBLE_UPPER)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N_INT, N_DOUBLE, INTEGER_UPPER(N_INT), &
    INTEGER_LOWER(N_INT)

INTEGER, INTENT(INOUT) :: CHROM_INT(N_INT), SEED

DOUBLE PRECISION, INTENT(IN) :: DOUBLE_LOWER(N_DOUBLE), &
    DOUBLE_UPPER(N_DOUBLE)

DOUBLE PRECISION, INTENT(INOUT) :: CHROM_DOUBLE(N_DOUBLE)

CHARACTER(LEN=30), INTENT(IN) :: MUT_TYPE

DOUBLE PRECISION :: CHROM_DOUBLE_NEW(N_DOUBLE), D_UP, D_LOW, RAN
INTEGER :: CHROM_INT_NEW(N_INT), LOC1, LOC2, N, N_UP, N_LOW

N=N_INT+N_DOUBLE
IF(TRIM(MUT_TYPE).EQ."UNIFORM")THEN
    LOC1=RANDOM_INTEGER(SEED,N,1)

    CHROM_INT_NEW=CHROM_INT
    CHROM_DOUBLE_NEW=CHROM_DOUBLE

    IF(LOC1.LE.N_INT) THEN
        N_UP=INTEGER_UPPER(LOC1)
        N_LOW=INTEGER_LOWER(LOC1)
        CHROM_INT_NEW(LOC1)=RANDOM_INTEGER(SEED,N_UP, N_LOW)
    ELSE
        LOC2=LOC1-N_INT
        D_UP=DOUBLE_UPPER(LOC2)
        D_LOW=DOUBLE_LOWER(LOC2)
        CHROM_DOUBLE_NEW(LOC2)=RANDOM_DOUBLE(SEED, D_UP, D_LOW)
    END IF

    CHROM_INT=CHROM_INT_NEW
    CHROM_DOUBLE=CHROM_DOUBLE_NEW
```

```
ELSE IF(TRIM(MUT_TYPE).EQ."SLIDING")THEN

    LOC1=RANDOM_INTEGER(SEED,N,1)

    CHROM_INT_NEW=CHROM_INT
    CHROM_DOUBLE_NEW=CHROM_DOUBLE

    RAN=RANDOM(SEED)

    IF(RAN.LE.0.5D0)THEN
        IF(LOC1.LE.N_INT) THEN
            N_UP=CHROM_INT(LOC1)
            N_LOW=INTEGER_LOWER(LOC1)
            CHROM_INT_NEW(LOC1)=RANDOM_INTEGER(SEED,N_UP, N_LOW)
        ELSE
            LOC2=LOC1-N_INT
            D_UP=CHROM_DOUBLE(LOC2)
            D_LOW=DOUBLE_LOWER(LOC2)
            CHROM_DOUBLE_NEW(LOC2)=RANDOM_DOUBLE(SEED,D_UP, D_LOW)
        END IF
    ELSE
         IF(LOC1.LE.N_INT) THEN
            N_UP=INTEGER_UPPER(LOC1)
            N_LOW=CHROM_INT(LOC1)
            CHROM_INT_NEW(LOC1)=RANDOM_INTEGER(SEED,N_UP, N_LOW)
        ELSE
            LOC2=LOC1-N_INT
            D_UP=DOUBLE_UPPER(LOC2)
            D_LOW=CHROM_DOUBLE(LOC2)
            CHROM_DOUBLE_NEW(LOC2)=RANDOM_DOUBLE(SEED,D_UP, D_LOW)
        END IF
    END IF

    CHROM_INT=CHROM_INT_NEW
    CHROM_DOUBLE=CHROM_DOUBLE_NEW

ELSE IF(TRIM(MUT_TYPE).EQ."BOUNDARY")THEN

    LOC1=RANDOM_INTEGER(SEED,N,1)

    CHROM_INT_NEW=CHROM_INT
    CHROM_DOUBLE_NEW=CHROM_DOUBLE

    RAN=RANDOM(SEED)

    IF (RAN.LE.0.5D0)THEN
        IF(LOC1.LE.N_INT)THEN
```

```
            CHROM_INT_NEW(LOC1)=INTEGER_UPPER(LOC1)
        ELSE
            LOC2=LOC1-N_INT
            CHROM_DOUBLE_NEW(LOC2)=DOUBLE_UPPER(LOC2)
        END IF
    ELSE
        IF(LOC1.LE.N_INT)THEN
            CHROM_INT_NEW(LOC1)=INTEGER_LOWER(LOC1)
        ELSE
            LOC2=LOC1-N_INT
            CHROM_DOUBLE_NEW(LOC2)=DOUBLE_LOWER(LOC2)
        END IF
    END IF

    CHROM_INT=CHROM_INT_NEW
    CHROM_DOUBLE=CHROM_DOUBLE_NEW

ELSE
    WRITE(*,*)"INVALID MUTATION TYPE: NO MUTATION PERFORMED"
END IF


END SUBROUTINE MUTATION

!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
! THIS SUBROUTINE RANDOMLY GENERATES A CHROMOSOME
!
SUBROUTINE CHROMOSOME_GENERATOR(SEED, N_INT,N_DOUBLE, CHROM_INT, &
    CHROM_DOUBLE, INT_UPPER, INT_LOWER, DOUBLE_LOWER, DOUBLE_UPPER)
IMPLICIT NONE
INTEGER, INTENT(INOUT) :: SEED, CHROM_INT(N_INT)

INTEGER, INTENT(IN) :: N_INT, N_DOUBLE, INT_UPPER(N_INT), &
    INT_LOWER(N_INT)

DOUBLE PRECISION, INTENT(INOUT) :: CHROM_DOUBLE(N_DOUBLE)

DOUBLE PRECISION, INTENT(IN) ::DOUBLE_LOWER(N_DOUBLE), &
    DOUBLE_UPPER(N_DOUBLE)
INTEGER :: I

DO I=1,N_INT,1
    CHROM_INT(I)=RANDOM_INTEGER(SEED, INT_UPPER(I), INT_LOWER(I))
END DO
```

```
DO I=1,N_DOUBLE,1
    CHROM_DOUBLE(I)=RANDOM_DOUBLE(SEED, DOUBLE_UPPER(I), &
        DOUBLE_LOWER(I))
END DO

END SUBROUTINE CHROMOSOME_GENERATOR

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE GENERATES THE INITIAL POPULATION
!
SUBROUTINE POPULATION_GENERATOR(N_POP, N_DOUBLE, N_INT, POP_DOUBLE, &
    POP_INT, SEED, INT_UPPER,INT_LOWER, DOUBLE_UPPER, DOUBLE_LOWER)
IMPLICIT NONE
INTEGER, INTENT(IN) :: N_POP, N_DOUBLE, N_INT, INT_UPPER(N_INT), &
    INT_LOWER(N_INT)
INTEGER, INTENT(INOUT) :: POP_INT(N_POP,N_INT), SEED
DOUBLE PRECISION, INTENT(INOUT) :: POP_DOUBLE(N_POP, N_DOUBLE)
DOUBLE PRECISION, INTENT(IN) :: DOUBLE_UPPER(N_DOUBLE), &
    DOUBLE_LOWER(N_DOUBLE)

INTEGER :: I, J

DO I=1,N_POP,1

    DO J=1,N_DOUBLE,1
        POP_DOUBLE(I,J)=RANDOM_DOUBLE(SEED, DOUBLE_UPPER(J), &
            DOUBLE_LOWER(J))
    END DO

    DO J=1,N_INT,1
        POP_INT(I,J)=RANDOM_INTEGER(SEED, INT_UPPER(J), INT_LOWER(J))
    END DO
END DO

END SUBROUTINE POPULATION_GENERATOR

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE GENERATES A RANDOM INTEGER WITHIN THE SPECIFIED
!  BOUNDS
!
FUNCTION RANDOM_INTEGER(SEED,UPPER,LOWER)
IMPLICIT NONE
```

```
INTEGER, INTENT(INOUT) :: SEED
INTEGER, INTENT(IN) :: UPPER, LOWER
DOUBLE PRECISION :: RAN
INTEGER :: RANDOM_INTEGER, DUM

IF(UPPER.EQ.LOWER) THEN
    RANDOM_INTEGER=UPPER
ELSE
    RAN=RANDOM(SEED)
    RANDOM_INTEGER=FLOOR(RAN*DBLE(UPPER+1-LOWER))+LOWER
END IF
!WRITE(*,*) RANDOM_INTEGER, LOWER, UPPER, SEED, RAN
END FUNCTION RANDOM_INTEGER

!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS SUBROUTINE GENERATES A RANDOM DOUBLE PRECISION NUMBER WITHIN
!  THE SPECIFIED BOUNDS
!
FUNCTION RANDOM_DOUBLE(SEED, UPPER,LOWER)
IMPLICIT NONE
INTEGER, INTENT(INOUT) :: SEED
DOUBLE PRECISION, INTENT(IN) :: UPPER, LOWER
DOUBLE PRECISION :: RAN, RANDOM_DOUBLE

    RAN=RANDOM(SEED)
    RANDOM_DOUBLE=RAN*(UPPER-LOWER)+LOWER

END FUNCTION RANDOM_DOUBLE

!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS  FUNCTION IS THE RAN FUNCTION FROM NUMERICAL RECIPES FOR
! FORTRAN 90.  IT USES THE RANDOM NUMBER GENERATOR OF PARK AND MILLER
!  COMBINED WITH A  MARSAGLIA SHIFT SEQUENCE.  THE PERIOD OF THIS
!  GENERATOR HAS A PERIOD OF ABOUT 3.1x10^18.  TO  INITIALIZE IT
!  IDUM SHOULD BE SET  TO  A NEGATIVE INTEGER VALUE.   AFTER THAT
!  IT'S VALUE SHOULDN'T BE CHANGED,EXCEPT TO REINITIALIZE.  THIS
!  FUNCTION IS TAKEN FROM THE NUMERICAL RECIPES FOR FORTRAN BOOK
FUNCTION RANDOM(IDUM)
IMPLICIT NONE
INTEGER, PARAMETER :: K4B=SELECTED_INT_KIND(9)
INTEGER(K4B), INTENT(INOUT) :: IDUM
DOUBLE PRECISION:: RANDOM
```

```
INTEGER(K4B), PARAMETER :: IA=16807, IM=2147483647,IQ=127773,IR=2836
DOUBLE PRECISION, SAVE :: AAM
INTEGER(K4B), SAVE :: IIX=-1, IIY=-1,KK

    IF(IDUM<=0 .OR. IIY<0) THEN
        AAM=NEAREST(1.D0,-1.D0)/IM
        IIY=IOR(IEOR(888889999,ABS(IDUM)),1)
        IIX=IEOR(777755555,ABS(IDUM))
        IDUM=ABS(IDUM)+1
    END IF

    IIX=IEOR(IIX,ISHFT(IIX,13))
    IIX=IEOR(IIX,ISHFT(IIX,-17))
    IIX=IEOR(IIX,ISHFT(IIX,5))
    KK=IIY/IQ
    IIY=IA*(IIY-KK*IQ)-IR*KK

    IF (IIY < 0) IIY=IIY+IM
    RANDOM=AAM*IOR(IAND(IM,IEOR(IIX,IIY)),1)

END FUNCTION RANDOM

!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
!*****************************************************************!
! THIS SUBROUTINE SORTS ARR FROM SMALLEST TO LARGEST, THE LOCATION OF
!  THE SORTED ORDER IS RETURN IN THE INDEX ARRAY.  HEAP SORT HAS AT
!  MOST AND ORDER OF O(NLOG(N)).  THIS ALGORITHM IS ADAPTED FROM THE
!  NUMERICAL RECIPES FOR FORTRAN BOOK.
!
SUBROUTINE HEAP_SORT(ARR, INDEX, N)
IMPLICIT NONE

INTEGER, INTENT(IN) :: N
INTEGER, INTENT(INOUT) :: INDEX(N)
DOUBLE PRECISION, INTENT(INOUT) :: arr(N)

INTEGER :: i, DUM_INT
DOUBLE PRECISION :: DUM_REAL

!n=size(arr)
do i=n/2,1,-1
call sift_down(i,n)
end do
do i=n,2,-1
!call swap(arr(1),arr(i))
    DUM_REAL=ARR(1)
```

```
    ARR(1)=ARR(I)
    ARR(I)=DUM_REAL
    DUM_INT=INDEX(1)
    INDEX(1)=INDEX(I)
    INDEX(I)=DUM_INT
call sift_down(1,i-1)
end do
CONTAINS

SUBROUTINE sift_down(l,r)
INTEGER, INTENT(IN) :: l,r
INTEGER :: j,jold
DOUBLE PRECISION :: A
INTEGER :: B
a=arr(l)
B=INDEX(L)
jold=l
j=l+l
do
if (j > r) exit
if (j < r) then
if (arr(j) < arr(j+1)) j=j+1
end if
if (a >= arr(j)) exit
    arr(jold)=arr(j)
    INDEX(JOLD)=INDEX(J)
jold=j
j=j+j
end do
arr(jold)=a
INDEX(JOLD)=B
END SUBROUTINE sift_down
END SUBROUTINE HEAP_SORT

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE IS THE WRAPPER FOR THE UNCMIN PACKAGE. THE UNCMIN
!  PACKAGES WAS MODIFIED TO PASS THE INTEGER CHROMOSOME/LENGTH,
!  INPUT ARRAY/SIZES, AND THE CONSTRAINT ARRAY G_CON/LENGTH.  THE
!  UNCMIN PACKAGE DOESN'T ACTUALLY USE THE CONSTRAINTS, BUT IT WAS
!  MODIFIED SO THE SAME COST FUNCTION CAN BE USED FOR ALL 3 NLP
!  SOLVERS.
!
SUBROUTINE UNCMIN_WRAPPER(N_DOUBLE, N_INT, N1, N2, ITMAX_NLP,  X0, &
    X, CHROM_INT, FITNESS, INPUT_ARRAY, G_CON, NCON)
IMPLICIT NONE
```

```
INTEGER, INTENT(IN) :: N_DOUBLE, N_INT, N1, N2, CHROM_INT(N_INT), NCON

INTEGER, INTENT(INOUT) :: ITMAX_NLP

DOUBLE PRECISION, INTENT(INOUT) ::  X(N_DOUBLE), X0(N_DOUBLE), &
    FITNESS, G_CON(NCON),INPUT_ARRAY(N1,N2)

INTEGER :: INFO

CALL UNCMIN(N_DOUBLE, N_INT, N1, N2, ITMAX_NLP, INFO, X0, X, &
    CHROM_INT, FITNESS, INPUT_ARRAY, G_CON, NCON)

    END SUBROUTINE UNCMIN_WRAPPER

!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
!**********************************************************************!
! THIS SUBROUTINE IS THE WRAPPER FOR THE CONMIN PACKAGE.  VERY LITTLE
!  MODIFICATION IS REQUIRED OF THE ORIGINAL CONMIN PACKAGE.  WITH
!  THIS VERSION OF CONMIN NO MORE THAN 2200 VARIABLES CAN BE
!  OPTIMIZED. THIS NLP SOLVER SHOULDN'T BE USED WITH PROBLEMS
!  CONTAINING MORE THAN 100-200 VARIABLES.  IF YOU HAVE THAT MANY
!  VARIABLES RUN TIMES WILL BE HIGH.
!
SUBROUTINE CONMIN_WRAPPER(NDV, N_INT, N1, N2, ITMAX, X0, X_OUT, &
                          CHROM_INT, FITNESS, INPUT_ARRAY, N_CON, &
                          DOUBLE_UPPER, DOUBLE_LOWER)
IMPLICIT NONE

INTEGER, INTENT (IN) :: NDV, N_INT, N1, N2, ITMAX, CHROM_INT(N_INT), &
    N_CON

DOUBLE PRECISION, INTENT(IN) :: DOUBLE_UPPER(NDV), DOUBLE_LOWER(NDV),&
    X0(NDV)

DOUBLE PRECISION, INTENT(INOUT) :: FITNESS, INPUT_ARRAY(N1,N2), &
    X_OUT(NDV)

DOUBLE PRECISION :: X(22*100), VUB(22*100), VLB(22*100)

INTEGER :: ISC(50*100), IER, IPRINT, NSIDE

X(1:NDV)=X0
VUB(1:NDV)=DOUBLE_UPPER
VLB(1:NDV)=DOUBLE_LOWER
! 1 = ENFORCE BOUNDARY CONDITION
```

```
! 0 = NO BOUNDARY CONDITIONS
NSIDE=1
! 0-5 GOES FROM PRINTING NOTHING TO PROGRESSIVELY MORE AND MORE
IPRINT=0
CALL CMINEX (X, VLB, VUB, ISC, NDV, N_CON, NSIDE, IPRINT, &
             ITMAX, IER, N1, N2, N_INT, CHROM_INT, FITNESS, &
             INPUT_ARRAY)

X_OUT=X(1:NDV)

END SUBROUTINE CONMIN_WRAPPER

!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
!*********************************************************************!
! THIS SUBROUTINE CONTAINS THE DRIVER FOR THE COBYLA OPTIMIZATION
!  PACKAGE.  THE ONLY MODIFICATIONS REQUIRED FOR THE COBYLA SOLVER IS
!  MODIFYING IT TO CALL THE COST FUNCTION AS REQURIED FOR THE GA-NLP
!  ALGORITHM
SUBROUTINE COBYLA_DRIVER(N, N_INT, N1, N2, x, CHROM_INT, FITNESS, &
    ARRAY, NCON, NLP_ITER_MAX )
IMPLICIT NONE

INTEGER, INTENT(IN) :: N, N_INT, N1, N2, CHROM_INT(N_INT), NCON, &
    NLP_ITER_MAX
DOUBLE PRECISION, INTENT(INOUT) :: FITNESS, X(N)

DOUBLE PRECISION, INTENT(IN) :: ARRAY(N1,N2)

DOUBLE PRECISION :: RHOBEG, RHOEND

INTEGER :: IPRINT, MAXFUN

MAXFUN=NLP_ITER_MAX
RHOBEG = 0.5D0
RHOEND = 1.D-6
IPRINT = 0

CALL COBYLA (N, NCON, X, RHOBEG, RHOEND, IPRINT, MAXFUN, N1, N2, &
    N_INT, CHROM_INT, ARRAY, FITNESS)

END SUBROUTINE COBYLA_DRIVER
```

BIBLIOGRAPHY

[1] Crocco, G.A. One-Year Exploration-Trip Earth-Mars-Venus-Earth. *VII International Astronautical Congress*, Sept. 1956.

[2] Ruppe, H.O. Minimum Energy Requirements for Space Travel. *X International Astronautical Congress*, 1959.

[3] Battin, R. Determination of Round-Trip Planetary Reconnaissance Trajectories. *Journal of the Aero/Space Sciences*, 26(9):545–567, Sept. 1959.

[4] Lawden, D.F. Interplanetary Rocket Trajectories. *Advances in Space Science*, 1, 1959.

[5] Breakwell, J.V. and Gillespie, R.W. and Ross, S. Researches in Interplanetary Transfer. *American Rocket Society Journal*, 31(2):201–208, 1959.

[6] M.A. Minovitch. The invention that opened the solar system to exploration. *Planetary and Space Science*, 58(6):885 – 892, 2010.

[7] C.F. Gauss and C.H. Davis. *Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections: A Translation of Gauss's "Theoria Motus." With an Appendix.* Little, Brown and Company, 1857.

[8] A. Hall. On a theorem of lambert's. *The Analyst*, 6(6):171–173, 1879.

[9] Lancaster, E.R. and Blanchard, R.C. A Unified Form of Lambert's Theorem. *NASA Technical Note, TN D-5368*, Sept. 1969.

[10] Gooding, R.H. On the Solution of Lambert's Orbital Boundary-Value Problem. *Royal Aerospace Establishment*, Apr. 1988.

[11] Loechler, L. An Elegant Lambert Algorithm for Multiple Revolution Orbits. Master's thesis, Massachusetts Institute of Technology, May 1988.

[12] Bate, R., Mueller, D., and White, J. *Fundamentals of Astrodynamics*. Ch. 5: Orbit Determination from Two Positions and Time, Dover Publications, Inc., 180 Varick Street, New York, NY, 1st edition, 1971.

[13] Battin, R. *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*. Ch. 7: Solving Lambert's Problem, AIAA Education Series, 1801 Alexadner Bell Drive, Reston, VA, Revised Edition edition, 1999.

[14] Vallado, D. *Fundamentals of Astrodynamics and Applications*. Ch. 7: Initial Orbit Determination, Microcosm Press, 401 Coral Circle, El Segundo, CA, 2nd edition, 2004.

[15] Arora, N. and Russell, R. GPU Accelerated Multiple Revolution Lambert Solver for Fast Mission Design. *AAS/AIAA Astrodynamics Specialist Conference*, AAS-10-198, Feb. 2010.

[16] The Portran Group. *CUDA Fortran Programming Guide and Reference*, 2014.

[17] Curtis, H. *Orbital Mechanics for Engineering Students*. Ch. 5: Preliminary Orbit Determination, Elsevier Butterworth-Heinemann, Linacre House, Jordan Hill, Oxford, 1st edition, 2005.

[18] Sun, F.T. On the Minimum Time Trajectory and Multiple Solutions of Lambert's Problem. *AAS/AIAA Astrodynamics Specialist Conference*, AAS-79-164, June 1979.

[19] Shen, H. and Tsiotras, P. Using Battin's Method to Obtain Multiple-Revolution Lambert's Solutions. *AAS/AIAA Astrodynamics Specialist Conference*, AAS-03-568, 2003.

[20] Gooding, R.H. A Procedure for the Solution of Lambert's Orbital Boundary-Value Problem. *Kluwer Academic Publishers*, Jan. 1990.

[21] Halley, E. Methodus nova accurata et facilis inveniendi radices aeqationum quarumcumque generaliter, sine praevia reductione. per edm. halley. *Philosophical Transactions (1683-1775)*, 18:136–148, 1694.

[22] Conway, B. An Improved Method Due to Laguerre for the Solution of Keplers Equation. *Celestial Mechanics*, 39:199–211, 1986.

[23] Prussing, J.E. and Conway, B.A. . *Orbital Mechanics*. Ch. 2: Position in Orbit as a Function of Time, Oxford University Press, 1993.

[24] Chobotov, V.A. . *Orbital Mechanics Third Edition*. Ch. 4: Positin and Velocity as a Function of Time, AIAA Education Series, 1801 Alexander Bell Drive, Reston, Virginia, 2002.

[25] Lawrence Livermore National Laboratory. *OpenMP*, 2002.

[26] Wie, B. *Space Vehicle Dynamics and Control*. American Institute of Aeronatics and Astronautics, Inc., Reston, VA 20191, 2nd edition, 2008.

[27] Wagner, S., Pitz, A., Zimmerman, D., and Wie, B. Interplanetary Ballistic Missile (IPBM) System Architecture Design for Near-Earth Object Threat Mitigation. *60th International Astronautical Congress* , IAC-09-D1.1.1, Oct. 2009.

[28] Wagner, S. and Wie, B. Design of Fictive Post-2029 Apophis Intercept Mission for Nuclear Disruption. *AIAA/AAS Astrodynamics Specialist Conference*, AIAA-2010-8375, Aug. 2010.

[29] Wagner, S. and Wie, B. A Crewed 180-Day Mission to Asteroid Apophis in 2028-2029. *60th International Astronautical Congress* , IAC-09-D2.8.7, Oct. 2009.

[30] Wagner, S. and Wie, B. Preliminary Design of a Crewed Mission to Asteroid Apophis in 2029-2039. *AIAA/AAS Astrodynamics Specialist Conference*, AIAA-2010-8374, Aug. 2010.

[31] Wie, B. and Dearborn, D. Earth-Impact Modeling and Analysis of a Near-Earth Object Fragmented and Dispersed by Nuclear Subsurface Explosions. *20th AAS/AIAA Space Flight Mechanics Meeting*, AAS-10-137, Feb. 2010.

[32] Kaplinger, B., Wie, B., and Dearborn, D. Preliminary Design of a Crewed Mission to Asteroid Apophis in 2029-2039. *AIAA/AAS Astrodynamics Specialist Conference*, AIAA-2010-79824, Aug. 2010.

[33] Korsmeyer, D., Landis, R., and Abell, P. Into the Beyond: A Crewed Mission to a near-Earth object. *Acta Astronautica*, 2008(63):213–220, April 2008.

[34] Landis, R. and Korsmeyer, D. and Abell, P. and Adamo, P. A Piloted Orion Flight to a Near-Earth Object: A Feasibility Study. *NASA*, 2001.

[35] Landis, R.n Abell, P.n Korsmeyer, D., Jone, T., and Adamo, D. . Piloted Operations at a Near-Earth Object (NEO). *Acta Astronautica*, 2009(65):1689–1967, June 2009.

[36] Gad, A. and Abdelkhalik, O. Hidden Genes Genetic ALgorithm for Multiple-Gravity-Assist Trajectories Optimization. *Journal of Spacecraft and Rockets*, 48(4), July-Aug. 2011.

[37] Abdelkhalik, O. and Gad, A. Dynamic-Size Multiple Populations Genetic Algorithm for Multigravity-Assist Trajectories Optimization. *Journal of Guidance, Control, and Dynamics*, 35(2), March-April 2012.

[38] Abdelkhalik, O. Autonomous Planning of Multigravity-Assist Trajectories with Deep Space Maneuvers Using a Differential Evolution Approach. *International Journal of Aerospace Engineering*, 2013, July 2013.

[39] Vinko, T. and Izzo, D. Global Optimisation Heuristics and Test Problems for Preliminary Spacecraft Trajectory Design. *ACT Tec. Rept. ACT-TNT-MAD-GOHTPPSTD, European Space Agency, the Advanced Concepts Team*, 2008.

[40] Englander, J.A. *Automated Trajectory Planning for Multiple Flyby Interplanetary Missions*. PhD thesis, University Of Illinois, Aug. 2012.

[41] Vavrina, M. *A Hybrid Genetic Algorithm Approach to Global Low-thrust Trajectory Optimization*. PhD thesis, Purdue University, 2008.

[42] Myatt, D. R., Becerra, V. M., Nasuto, S. J. and Bishop, J. M. Advanced Global Optimisation for Mission Analysis and Design. Technical Report 03-4101a, ESA Ariadna, 2004.

[43] Pascale, P. De and Vasile, M. Preliminary Design of Low-Thrust Multiple Gravity-Assist Trajectories. *Journal of Spacecraft and Rockets*, 43(5):1065–1076, Sept. 2006.

[44] Eberhart, R.C. and Kennedy, J. A New Optimizer Using Particle Swarm Theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[45] Storn, R. and Price, K. Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[46] R. Gehrke. *First-principles basin-hopping for the structure determination of atomic clusters*. PhD thesis, Berlin, Freie Universitat Berlin, Diss., 2009, 2009.

[47] Theodore W. Manikas and James T. Cain. Genetic algorithms vs. simulated annealing: a comparison of approaches for solving the circuit partioning problem, 1996.

[48] Kim, M. *Continuous Low-Thrust Trajectory Optimization: Techniques and Applications*. PhD thesis, Virginia Polytechnic Institute and State University, April 2005.

[49] Yam, C.H. and Di Lorenzo, D. and Izzo, D. Constrained global optimization of low-thrust interplanetary trajectories. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7, July 2010.

[50] McConaghy, T. and Longuski, J. Parameterization Effects on Convergence when Optimizing a Low-Thrust Trajectory with Gravity Assists. *AIAA/AAs Astrodynamics Specialist Conference and Exhibit*, Aug. 2004.

[51] Lantoine, G. *A Methodology for RObust Optimization of Low-Thrust Trajectories in Multi-Body Environments*. PhD thesis, Georgia Institute of Technology, December 2010.

[52] Conway, B. *Spacecraft Trajectory Optimization*. Ch. 3: Spacefraft Trajectory Optimization Using Direct Transcription and Nonlinear Programming, Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013-2473, USA, First edition, 2010.

[53] Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, First edition, 1989.

[54] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, First edition, 1992.

[55] Syswerda, G. Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[56] Schnabel, R., Koontz, J., and Weiss, B. A Modular System of Algorithms for Unconstrained Minimization. *University of Colorado at Boulder: Department of Computer Science*, 1985.

[57] Kahaner, D., Moler, C., and Nash, S. *Numerical Methods and Software*. Prentice Hall Series in Computational Mathematics, Englewood Cliffs, New Jersey 07632, 2nd edition, 1989.

[58] Broyden, C.G. The convergence of a class of double rank minimization algorithms: 2. The new algorithm. *Journal of the Institute of Mathematics and its Applications*, 6:76–231, 1970.

[59] Fletcher, R. A New Approach to Variable Metric Algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[60] Goldfarb, D. A family of variable metric methods derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[61] Shanno, D.F. Conditioning of quasi-Newton methods for function mini- mization. *Mathematics of Computation*, 24:647–650, 1970.

[62] Vanderplaats, G.N. Conmin User's Manual . Technical Report X-62282, NASA Technical Memorandum , 1978.

[63] Vanderplaats, G. N., and Moses, F. Structural Optimization by Methods of Feasible Directions. *National Symposium on Computerized Structural Analysis and Design*, March 1972.

[64] Powell, M.J.D. A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In *Advances in Optimization and Numerical Analysis*, volume 275 of *Mathematics and Its Applications*, pages 51–67. Springer Netherlands, 1994.

[65] Powell, M.J.D. A View of Algorithms for Optimization without Derivatives. Technical Report DAMTP 2007/NA03, Cambridge University, 2007.

[66] Molga, M. and Smutnicki, C. Test Functions for Optimization Needs. 2005.

[67] Rastrigin, L.A. Extremal control systems. *Theoretical Foundations of Engineering Cybernetics Series*, 1974.

[68] Bäck, T. *Evolutionary ALgorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Ch. 3: Artificial Landscapes, Oxford University Press, Inc., 198 Madison Ave., New York, NY, 1996.

[69] Englander, J., Conway, B., and Williams, T. Optimal Autonomous Mission Planning Via Evolutionary Algorithms. *21th AAS/AIAA Space Flight Mechanics Meeting*, AAS-11-159, Feb. 2011.

[70] Hohmann, W. The Attainability of Heavenly Bodies (Translated from German). Technical report, NASA Technical Translation, Nov. 1960.

[71] Goddard, R.H. Three Astronautical Pioneers - 1 Robert H. Goddard an autobiography. *Astronautics*, Apr. 1959.

[72] Tsiolkovskii, K.E. and Petroff, A.N. Three Astronautical Pioneers - 2 K.E. Tsiolkovskii an autobiography. *Astronautics*, May 1959.

[73] D'Amario, L., Bright, L. and Wolf, A. Galileo Trjectory Design. *Space Science Review*, 60:23–78, 1992.

[74] Peralta, F. and Flanagan, S. Cassini Interplanetary Trajectory Design. *Control Eng. Practice*, 3(11):1603–1610, 1995.

[75] Vasile, M. and De Pascale, P. On the Preliminary Design of Multiple Gravity-Assist Trajectories. *Journal of Spacecraft and Rockets*, 43(4):794–805, 2009.

[76] Wagner, S., Kaplinger, B., and Wie, B. GPU Accelerated Genetic Algorithm for Multiple Gravity-Assist and Impulsive ΔV Maneuvers. *AIAA/AAS Guidance Navigation and Control Conference*, AIAA 2012-4592, Aug. 2012.

[77] Wagner, S. and Wie, B. Low-Thrust Trajectory Optimization for Asteroid Exploration, Redirct, and Deflection Mission. *24th AAS/AIAA Spaceflight Mechanics Meeting*, AAS 14-283, Jan. 2014.

[78] Qadir, K. Mult-gravity Assist Design Tool for Interplanetary Trajectory Optimisation. Master's thesis, Lulea University of Technology, 2009.

[79] Davis, K. and Parker, J. . Constructing Resonant Orbits. University of Colorado Boulder:ASEN 5519-Interplanetary Mission Design, 2012.

[80] Morrison, D. The Spaceguard Survey: Report of the NASA International Near-Earth-Object Detection Workshop. *NASA STI/Recon Technical Report N*, 92:34245, Jan. 1992.

[81] Pitz, A., Kaplinger, B., and Wie, B. Preliminary Design of a Hypervelocity Nuclear Interceptor Spacecraft for Optimal Disruption/Fragmentation of NEOs. *22nd AAS/AIAA Space Flight Mechanics Meeting*, AAS-12-225, Jan. 2012.

[82] Vardaxis, G., Pitz, A., and Wie, B. Conceptual Design of Planetary Defense Technology Demonstration Mission. *22nd AAS/AIAA Space Flight Mechanics Meeting*, AAS-12-128, Jan. 2012.

[83] Winkler, T., Wagner, S., and Wie, B. Optimal Target Selection for a Planetary Defense Technology (PDT) Demonstration Mission. *22nd AAS/AIAA Space Flight Mechanics Meeting*, AAS-12-226, Jan. 2012.

[84] Carnelli, I., Gálvez, A., and Izzo, D. Don Quijote: A NEO Deflection Precursor Mission. *2006 NASA Near-Earth Object Detection and Threat Mitigation Workshop*, June 2006.

[85] Cano, J., Sánchez, M., and Carnelli, I. Mission Analysis for the Don Quijote Phase-A Study. *Proceedings of the 20th International Symposium on Space Flight Dynamics*, Sept. 2007.

[86] Gálvez, A. and Carnelli, I. ESA's Don Quijote Mission: an Opportunity for the Investigation of an Artificial Impact Crater on an Asteroid. *25th International Symposium on Space Technology and Science*, 2006.

[87] Fujiwara, A., Kawaguchi, J., Yeomans, Et Al. The Rubble-Pile Asteroid Itokawa as Observed by Hayabusa. *Science*, 312(5778):1330–1334, 2006.

[88] Wagner, S., Winkler, T., and Wie, B. Analysis and Selection of Optimal Targets for a Planetary Defense Technology Demonstration Mission. *AIAA/AAS Guidance Navigation and Control Conference*, AIAA 2012-4874, Aug. 2012.

[89] Sims, J.A. and Flanagan, S.N. Preliminary Design of Low-Thrust Interplanetary Missions. *AAS/AIAA Astrodynamics Specialist Conference*, AAS 99-338, Aug. 1999.

[90] Keck Institute for Space Studies. Asteroid Retrieval Reasibility Study. Technical report, Keck Institute for Space Studies, California Institute of Technology, Jet Propulsion Laboratory, Apr. 2012.

[91] Szabo, J., Hruby, V., Byrne, L., Tedrake, R., Kolencik, G., Kamhawi, H., and Haag, T. A Commercial One Newton Hall Effect Thruster for High Power In-Space Missions. *47th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, AIAA 2011-6152, July 2011.

[92] Brophy, J.R. and Muirhead, B. Near-Earth Asteroid Retrieval Mission (ARM) Study. *33rd International Electric Propulsion Conference*, IEPC-2013-82, Oct. 2013.

[93] Meeus, J. *Astronomical Algorithms*. Ch. 7: Julian Day, Willmann-Bell, Inc., Richmond, VA, 1991.