The Pennsylvania State University

The Graduate School

Department of Aerospace Engineering

SOLAR SAIL TRAJECTORY OPTIMIZATION USING COLLOCATION AND NSGA II

A Thesis in

Aerospace Engineering

by

Elizabeth Catherine Davis

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2008

The thesis of Elizabeth Davis was reviewed and approved* by the following:

Robert G. Melton
Professor of Aerospace Engineering
Thesis Advisor

David B. Spencer
Associate Professor of Aerospace Engineering

George A. Lesieutre
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Solar sail spacecraft rely solely on solar radiation pressure for propulsion. Since the acceleration received by radiation pressure increases as the spacecraft nears the Sun, missions to inner solar system planets are commonly proposed. Finding optimal trajectories for such missions can be difficult because the equations of motion must be solved numerically and the optimization problem includes many nonlinear constraints. One technique that has good shown good results is direct collocation with non-linear programming (DCLNP). The method sets up the problem as a non-linear programming problem which is traditionally solved with a method that requires gradient information. However, such solvers generally need a good initial guess to converge. This research found that the trajectory optimization problem formulated as a DCLNP and optimized with the evolutionary algorithm, NSGA II, can be used to create sub-optimal trajectories for the solar sail spacecraft. Although DCNLP includes a number of nonlinear algebraic constraints, these cannot be satisfied exactly with an evolutionary algorithm. Therefore, Pareto multi-objective optimization was used to minimize error in these constraints, while also minimizing the time-of-flight. The resulting trajectories could then be used as initial guesses for more traditional NLP solvers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Melton, for his help throughout the thesis process. I would also like to acknowledge my parents, family, and the surface of the Earth for their support throughout my education.

CHAPTER 1

INTRODUCTION

Solar sails are a type of spacecraft that rely solely on radiation pressure from the Sun for propulsion, unlike traditional spacecraft. The force from solar radiation pressure is modeled as a perturbation to the central-body force due to the gravitational attraction of the Sun. Optimal trajectories for this type of spacecraft can not be solved for analytically; such solutions must be found using computational methods. These can be indirect, which uses calculus of variations, or direct, which optimizes the state variables directly. A common direct method employed in trajectory optimization is direct collocation with non-linear programming (DCNLP) which approximates the trajectory to minimize computation time and then transforms the problem into a non-linear programming (NLP) problem. This method has been used with gradient-based solvers in the past with good results. However, these solvers require a good initial guess to converge. By using an additional optimization technique, such as an evolutionary algorithm (EA) an approximate solution can be obtained in a fairly short amount of time. In this thesis, a freely available EA, the Non-Dominated Sorting Genetic Algorithm II (NSGA II), is used with the DCNLP method to obtain suboptimal trajectories. These solutions can then be fed to a traditional NLP solver as an initial guess.

The original application of DCNLP method was finite-thrust applications [1]. It was later used on low-thrust trajectories for spacecraft powered by electric propulsion [2] [3] as well as solar sail trajectories [4] with good results using traditional gradient-based NLP solvers to search for optimal trajectories.

More recent research has focused on using evolutionary techniques. A Pareto optimal, multi-objective genetic algorithm (NSGA) hybridized with a gradient based search method was

shown to produce novel results; however, it was found to be computationally expensive [5]. Evolutionary algorithms have also been used to produce initial guesses for traditional gradient based NLP solvers [6][7]. The NSGA-II algorithm that is used in this work has been used for optimizing control laws for coplanar orbit transfers. It was compared with its predecessor NSGA and was found to perform better by covering more of the Pareto front and converging in a smaller number of generations [8].

Several other evolutionary techniques have also been employed for the trajectory optimization problem. One is evolutionary branching which combines an EA to search for the possible presence of an optimum and a systematic search algorithm which breaks the problem domains into sub domains to search for the global optimum [9]. Another method has employed adaptive machine learning called evolutionary neurocontrol to find near-optimal trajectories [10]. Ant colony algorithms have been successfully used with impulsive inter-planetary transfer test problems, but more work is needed for a full trajectory optimization [11].

Chapter 2 contains an overview of solar sails and derivation of the equations of motion. The method of direct collocation employed in the thesis is described in Chapter 3. A survey of basic EA principles and outline of the NSGA-II algorithm used to carry out the optimization is in Chapter 3. The results of the optimization simulations are given in Chapter 5. The last chapter contains conclusions and suggestions for further work.

CHAPTER 2

SOLAR SAIL SPACECRAFT

2.1 Introduction

Solar sail spacecraft differ from traditional spacecraft in that they rely solely on radiation pressure from solar photons for acceleration instead of chemical propulsion. Since the force from radiation pressure is very small, a solar sail spacecraft must have a much larger surface area than conventional spacecraft to receive a serviceable amount of acceleration from it. The mass-to-area ratio, $\sigma = m/A$, serves as an import design characteristic that, in part, determines the acceleration for a particular sail [12]. In this thesis, the performance of sails with σ values of 0.002 kg/m^2 to 0.2 kg/m^2 is studied. This corresponds to sail areas of 2,500 m^2 to 250,000 m^2 for a 500 kg spacecraft. It is expected that the first generation of solar sails actually deployed will have values of σ around 0.013 kg/m^2 [15].

The sail consists of a thin reflecting film, usually aluminum on a substrate like Kapton or Mylar. Aluminum is not a perfect reflector, so it will not produce as much acceleration as an ideal sail. This is another design factor that must be taken into account for a realistic model of a solar sail's performance. It also implies that the sail will absorb some energy in the form of heat. On missions to near-Sun regions, this can be a serious design constraint [13]. Many configurations of sails have been proposed. They all consist of large areas of sail material with some sort of stabilization such as booms or inertia due to rotation [12], [14].

The idea of using electromagnetic radiation as a means of propulsion was first proposed in the 1920's, but no real studies were undertaken until the 1950's. Research into solar sails is currently ongoing. There have been small scale tests of solar sails up 20 m on a side in vacuum

chambers, [15], but currently there have been no actual space tests of solar sail spacecraft to date. One space test was attempted with the Cosmos 1 spacecraft, but did not get into orbit due to a failure with the launch vehicle [16].

Many possible future missions have been proposed [14]. The planet Mercury is a popular target because traditional spacecraft require a large ΔV to reach it, but is well suited for investigation with a solar sail spacecraft. They benefit from the increase in the radiation pressure nearer to the Sun [19] [20]. As with most missions, its duration can be further shortened by adding gravity assist maneuvers, specifically at Venus [13]. Even missions to the outer solar system have even been proposed, some that include maneuvers that involve first nearing the Sun to gain more acceleration [18], but the inner Solar System planets and Mars [8][19] [12] are the most common targets.

2.2 Basic Principles

The energy of a single photon can be expressed as h/v where h is Planck's constant ($6.626\times10^{-34}\,\mathrm{J\cdot s^{-1}}$) and v is the frequency. From the special theory of relativity comes the idea that photons, although massless, can impart momentum to an object because they have energy. For a relativistic particle, the relationship between energy, E, and momentum, p, is given by

$$E^2 = p^2c^2 + m^2c^4 \qquad 2.1$$

where c is the speed of light in a vacuum, 2.998×10^8 m/s.

A star's luminosity, L_s, is the power it outputs at its surface. For the Sun, the value of L_s is taken to be 3.827×10^{26} W. As the distance from the star increases, its power/area output decreases by an inverse square relationship. The flux at a distance r from a star is

$$\Phi = \frac{L_s}{4\pi r^2} \qquad 2.2$$

Flux also can be expressed as $\Phi = \frac{\Delta E}{A\Delta t}$ or the amount of energy transferred per area per time. Using equation 2.1, it can be written in terms of momentum transferred, Δp,

$$\Phi = \frac{\Delta pc}{A\Delta t} \qquad 2.3$$

Rearranging this equation, the magnitude of the force due to the radiation pressure can be found

$$F_{rp} = \frac{dp}{dt} = \frac{\Phi A}{c} \qquad 2.4$$

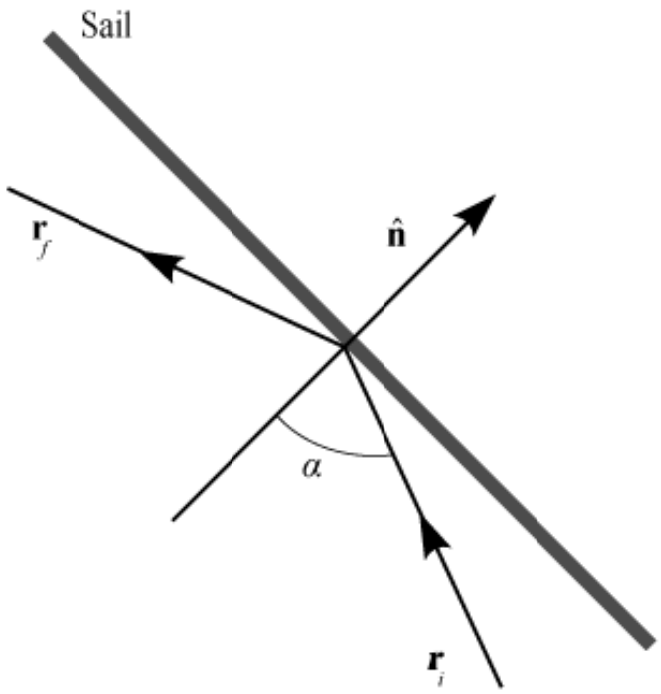

Figure 2-1: Forces on Solar Sail

To find the direction of the force, let $\hat{\mathbf{n}}$ be the unit vector normal to the sail area A. The force the sail will experience will be $\mathbf{f}_i = F_{rp}\left(\mathbf{r}_i \cdot \hat{\mathbf{n}}\right)^2 \hat{\mathbf{n}}$ due to the projection of $\hat{\mathbf{n}}$ in the $\boldsymbol{r_i}$ direction. There will also be a force, equal in magnitude, from the reflected photons in the opposite direction. Combined, this gives

$$\mathbf{F}_{rp} = 2F_{rp}\left(\mathbf{r}_i \cdot \hat{\mathbf{n}}\right)^2 \hat{\mathbf{n}} \qquad 2.5$$

for an ideally reflective solar sail. To increase the accuracy of the force model by taking into account the reflection of non-ideal materials, the factor of two is replaced with the correction term, $\left(1 + C_{rf}\right)$ where C_{rf} is the coefficient of reflection of the material.

By substituting equations 2.2 and 2.4 into equation 2.5, the an expression acceleration due to solar radiation pressure is simply

$$\mathbf{a}_{rp} = \frac{L_s}{2\pi\sigma c} \cdot \frac{1}{r^2}\left(1 + C_{rf}\right)\left(\mathbf{r}_i \cdot \hat{\mathbf{n}}\right)^2 \hat{\mathbf{n}} \qquad 2.6$$

This express allows the acceleration to be written in terms of the sail orientation.

2.3 Equations of Motion

The radiation pressure derived above is treated as a perturbation to the central gravitational force. So the total acceleration, $\ddot{\mathbf{r}}_T$, experienced by the sail becomes

$$\ddot{\mathbf{r}}_T = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_{rp} \qquad 2.7$$

To simplify calculations, a rotating cylindrical coordinate system (shown in Figure 2-2) is used. It is defined so that the radial component always points along straight line connecting the Sun and the spacecraft, and the angular component is allowed to rotate with the spacecraft. The non-planar component represents how far above the Ecliptic plane the spacecraft is located. The sail's orientation is described by the in-plane pitch angle, α, and the out-of-plane pitch angle δ.

The acceleration experienced by the sail must be transformed to account for the rotation of the system with

$$\mathbf{a}_{rot} = \ddot{\mathbf{r}}_T - \dot{\boldsymbol{\omega}} \times \mathbf{r} - \boldsymbol{\omega} \times \left(\boldsymbol{\omega} \times \mathbf{r}\right) - 2\boldsymbol{\omega} \times \mathbf{v}_r \qquad 2.8$$

where $\boldsymbol{\omega}$ is the vector rotation rate of the frame and $\mathbf{v}_r$ the relative motion of the spacecraft to the frame. The rotation occurs around the $\hat{\mathbf{z}}$ axis so that the $\hat{\boldsymbol{\theta}}$ terms of $\mathbf{r}$ are zero.

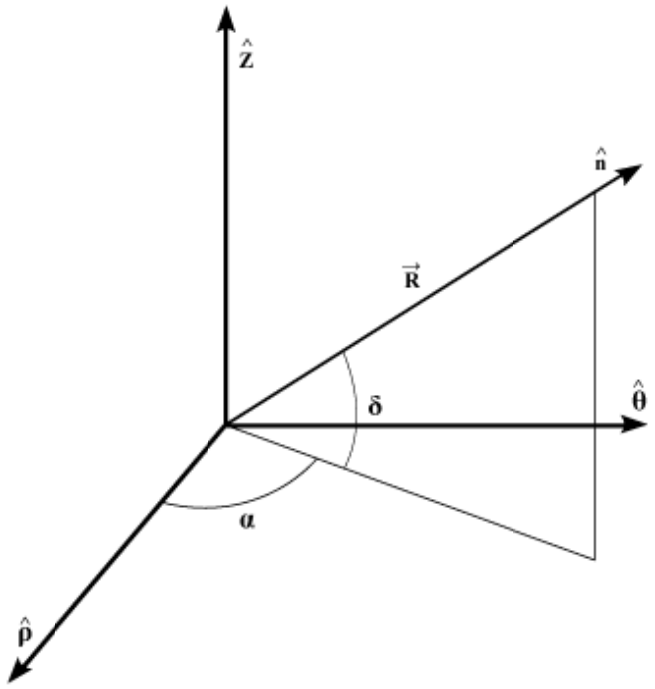

Figure 2-2: Coordinate System

Thus, Equation 2.8 reduces to

$$\mathbf{a}_{rot} = \ddot{\mathbf{r}}_T + \rho\omega^2\hat{\boldsymbol{\rho}} - \left(2\dot{\rho}\omega + \rho\ddot{\omega}\right)\hat{\boldsymbol{\theta}} \qquad 2.9$$

If $\mathbf{R} = f(\rho,\theta,z)$ is the position vector of a spacecraft as shown in Figure 2-2, the components of the sail normal vector can be found as a function of α and δ.

$$\hat{\mathbf{n}} = \frac{\mathbf{R}}{|\mathbf{R}|} = \cos(\delta)\cos(a)\hat{\boldsymbol{\rho}} + \cos(\delta)\sin(a)\hat{\boldsymbol{\theta}} + \sin(\theta)\hat{\mathbf{z}} \qquad 2.10$$

Replacing $\mathbf{r}_i$ with $\mathbf{R}$ and $\hat{\mathbf{n}}$ with Equation 2.10, the term $\left(\mathbf{r}_i \cdot \hat{\mathbf{n}}\right)^2$ in Equation 2.6 becomes

$$\lambda = \left(\mathbf{R}\cdot\hat{\mathbf{n}}\right)^2 = \left[\frac{\rho}{r}\cos(\delta)\cos(\alpha) + \frac{z}{r}\sin(\delta)\right]^2 \qquad 2.11$$

where $r = \sqrt{\rho^2 + z^2}$. Substituting Equations 2.10, 2.11, and 2.6 into Equation 2.7 gives

$$\ddot{\mathbf{r}}_T = \left(\frac{\Omega}{r^2}\lambda\cos(\delta)\cos(\alpha) - \frac{\mu}{r^3}\rho\right)\hat{\boldsymbol{\rho}} + \left(\frac{\Omega}{r^2}\lambda\cos(\delta)\sin(\alpha)\right)\hat{\boldsymbol{\theta}} + \left(\frac{\Omega}{r^2}\lambda\sin(\delta) - \frac{\mu}{r^3}z\right)\hat{\mathbf{z}} \qquad 2.12$$

where

$$\Omega = \frac{L_s}{4\pi c}\cdot\frac{A}{m}\cdot(1 + C_{rf}) \qquad 2.13$$

The final equations of motion are then found by substituting Equation 2.12 into 2.9 to give

$$\begin{aligned}
\dot{\rho} &= v_\rho \\
\dot{\theta} &= \omega \\
\dot{z} &= v_z \\
\dot{v}_\rho &= \frac{\Omega}{r^2}\lambda\left(\cos(\delta)\cos(\alpha)\right) - \frac{\mu}{r^3}\rho + \omega^2\rho \\
\dot{\omega} &= \frac{1}{\rho}\left[\frac{\Omega}{r^2}\lambda\left(\cos(\delta)\sin(\alpha)\right) - 2v_\rho\omega\right] \\
\dot{v}_z &= \frac{\Omega}{r^2}\lambda\sin(\delta) - \frac{\mu}{r^3}z
\end{aligned} \qquad 2.14$$

CHAPTER 3

COLLOCATION FOR TRAJECTORY OPTIMIZATION

Trajectory optimization for solar sail spacecraft requires solving a constrained, optimal control problem. Since they can be difficult to solve directly, various methods have been devised to simplify the problem [20]. The method that has been employed here is direct collocation with non-linear programming (DCNLP) as outlined by Enright and Conway originally for finite thrust trajectories [1]. DCNLP transforms the optimal control problem into a non-linear programming (NLP) problem. Solving the equations of motion directly is computationally expensive; instead, an approximation to the true trajectory is used. The trajectory is divided into a finite number of segments and the governing differential equations of motion are solved implicitly for each segment. For each segment, the difference between certain derivative properties in the exact and approximate solutions (known as the defect) is minimized, while also minimizing a cost function, to produce an optimal trajectory.

Initially, the trajectory is divided into a number of equal time segments joined at nodes. More nodes allows for a more accurate approximation because each approximated segment then covers a smaller portion of the true trajectory. However, this increases the problem size making the optimization more difficult since the number of variables and defects depends on the number of nodes.

The true trajectory along each segment is approximated by a Hermite cubic polynomial. If the derivative of the polynomial has the same value as that obtained from the equations of motions calculated at the center of a segment (i.e. the defect at the point is zero) then the polynomial is considered an accurate approximation of the true trajectory. The state vector of the polynomial at the center of the segment is given by

$$\mathbf{x}_c = \frac{1}{2}\left(\mathbf{x}_r + \mathbf{x}_l\right) + \frac{T}{8}\left(f(\mathbf{x}_l) - f(\mathbf{x}_r)\right) \tag{3.1}$$

where T is the time interval of the segment, $\mathbf{x}_l$ and $\mathbf{x}_r$ are the state vectors at the respective left and right nodes and $f(\mathbf{x}_l)$ and $f(\mathbf{x}_r)$ are the equations of motion evaluated at those points. The derivative of the interpolated polynomial state vector at the center is then

$$\dot{\mathbf{x}}_c = -\frac{3}{2T}\left(\mathbf{x}_l - \mathbf{x}_r\right) - \frac{1}{4}\left(f(\mathbf{x}_l) - f(\mathbf{x}_r)\right) \tag{3.2}$$

The control vector, $\boldsymbol{u}$, is assumed to be linear across a segment and is given by

$$\mathbf{u}_c = \frac{1}{2}\left(\mathbf{u}_l + \mathbf{u}_r\right) \tag{3.3}$$

Thus the defect, d, for a segment is merely

$$d = f\left(\mathbf{x}_c, \mathbf{u}_c\right) - \dot{\mathbf{x}}_c \tag{3.4}$$

The defect is a performance measure of how well the polynomial satisfies the equations of motion. As the approximation becomes more accurate, the defect goes toward zero.

After the trajectory is discretized, it is converted into a NLP problem. A NLP state vector, **X**, is created by combining the state and control vectors at each node and the total time of the trajectory.

$$\mathbf{X}^T = \left[\mathbf{x}_1{}^T, \mathbf{u}_1{}^T, \mathbf{x}_2{}^T, \mathbf{u}_2{}^T, \ldots, \mathbf{x}_N{}^T, \mathbf{u}_N{}^T, \tau\right] \tag{3.5}$$

where N is the total number of nodes and τ the time of flight. Another vector of constraints, **C**,

$$\mathbf{C}^T = \left[d_1, d_2, \ldots, d_{N-1}, \mathbf{w}^T\right] \tag{3.6}$$

containing the defects and a vector of problem constraints, **w**, is traditionally created and used in the optimization. To solve the optimization problem with an EA, the actual formulation of the optimization problem is modified as described in Chapter 5.

CHAPTER 4

EVOLUTIONARY ALGORITHMS

The evolutionary algorithm employed to optimize the collocated trajectory problem was the Non-Dominated Sorting Genetic Algorithm II (NSGA II) [21]. It is an EA used to find Pareto-optimal solutions to multi-objective optimization problems [22]. Traditional NLP solvers require a good initial guess to converge to an optimal solution, and even then, they might fail to converge for especially large problems [20]. In this case, EA's are useful because they do not need a good initial guess to find an optimal solution, although using one can help speed convergence. They also have been able to find novel trajectories that traditional methods miss [5] and are also easy to implement because they need no prior knowledge of the problem.

4.1 Evolutionary Algorithms

Evolutionary Algorithms are stochastic algorithms that use evolutionary principles on a population of possible solutions to search for optima. Each member of the population encodes for a solution vector, called a chromosome. In this case, the chromosome is a vector of real values since the problem space is continuous. To search for solutions, EAs use three major operators: selection, crossover, mutation [23].

Selection is the operator that guides the search of an EA toward solutions with better fitness values (in EA terminology, *fitness value* is equivalent to the term *cost function* employed in nonlinear optimization). In general, it replaces poor solutions with better ones. There are several selection strategies EAs commonly employ. The one that NSGA II uses is binary tournament selection. Two members of the population are selected, and the better solution

member is allowed to reproduce and replace the poorer solution. To help speed convergence, elitism is also employed. Elitism is a strategy to preserve good solutions from the previous generations by allowing them to survive into the current generation.

The recombination operator creates new solutions from two or more parent solutions. The type of recombination used here is simulated binary crossover (SBX). It is suppose to simulate single point crossover for binary strings where a point on the parent chromosomes is selected and the data after that point are switched between the two parents to produce offspring solutions. To apply the idea to a real coded problem, a probability distribution is created based on the spread of the parent solutions that is used to generate the offspring. Two offspring solutions are generated for each pair of two parent solutions to eliminate bias toward a particular parent. [24].

Mutation is another operator that generates new solutions. It does so by randomly perturbing a parent solution to create offspring. The amount the solution is perturbed and how much mutation takes place are user-defined inputs.

Evolutionary algorithms are basically loop programs. An initial population is randomly generated. Next the mutation and recombination operators are applied to generate new solutions. Then, the selection operator drives the EA toward solutions with better fitness values. The process continues until some specified stopping criterion is reached. In most cases, the longer an EA is allowed to run, the better the resulting solutions will be.

4.2 Pareto Optimality

Multi-objective optimization problems can be difficult because many have conflicting objectives that need to be balanced. One way to deal with this is Pareto optimality. A solution is considered Pareto optimal if there is no other solution that will increase the quality of one of the objective functions without decreasing the quality of another. Better solutions are considered to

dominate less optimal solutions. The result is a set of non-dominated solutions, called a Pareto front, that allow the user to find a compromise between conflicting objectives depending on the problem. Using Pareto optimality is more efficient than the traditional approach. It allows the Pareto optimal set of solutions to be found in a single run of the EA. Traditionally, the problem would need to be transformed from multi-objective to single-objective and each run of the EA would return only a single solution on the Pareto front, so to find the entire optimal set, it would need to be to be run multiple times.

4.3 Constraints

Problem constraints are another difficulty, like multi-objective problems, that occur with optimization problems. They show up due to the physical realities of a problem and are used to reject infeasible solutions. Constraints traditionally are implemented so as to penalize solutions that are infeasible. Depending on how they are implemented, they can restrict the EA when searching for a solution and might cause it to fail to find the optimum. These can also be difficult to apply because the penalty values for infeasible solutions need to be tuned for the best performance of the EA.

NSGA II uses a different method to implement constraints; essentially it treats them as another set of objective values. The total amount each solution violates the constraints is calculated and used when ranking solutions based on dominance. If two solutions have the same objective values, but different constraint violations, the one that violates the constraints the least will dominate the other solution. Thus, all feasible solutions will be higher ranked than any infeasible solution, regardless of objective values [22].

The trajectory optimization problem studied in this thesis has two sets of constraints, one of equality constraints and the other of inequality constraints. There are equality constraints on

the endpoints. The spacecraft needs to match the speed and velocity of the planet so it can enter into orbit around it. The defect values are also equality constraints; they need to equal zero for the approximation to be the true trajectory. The rest of the constraints were inequality constraints implemented on the position variables so the spacecraft would go smoothly toward the target planet.

4.4 NSGA II

NSGA II was first proposed in 2000 and has since been used in a variety of engineering problems. Based on an earlier EA, NSGA, it improves upon the original by having fewer user-specified parameters, employing elitism, and having a faster solution sorting function. Currently, it has not yet been used in a direct trajectory optimization problem, although its predecessor has [5]. It has been used to optimize control laws for orbit transfers [8]. NSGA II was used to minimize the defect values and the time of flight for a solar sail spacecraft trajectory formulated as a DCNLP problem.

Besides the standard operators, NSGA II employs another operator, the crowding comparison operator. The purpose of this operator is guide selection toward solutions that uniformly cover the Pareto optimal front. Crowding distance is an estimate of the density of solutions along a Pareto front. Solutions that are farther apart will have a larger crowding distance. It is used with domination rank by the operator to select solutions. Solutions with a lower rank (less dominated), are preferred to those with a higher rank. If the ranks are equal, the solution in a less crowded portion of the front is favored. After the initial generation, it is used as the criterion for selection, instead of just fitness or domination rank which would not preserve diversity in the population.

The first generation of NSGA II creates a random parent population which is ranked according to dominance. Then selection, recombination and mutation operators are used to create a population of offspring. In succeeding generations, elitism is employed by creating a population that is a union of both the parents and offspring solution sets. It is twice the size of the initial population. This new population is sorted based on rank allowing the best solutions of the current and previous generations to compete. Then the crowding comparison operator is used to fill the parent population of the next generation. The selection, mutation, and recombination operators are used on this new parent population to create the offspring for the current generation. This process then continues until the program reaches the specified number of generations.

The standard performance measures for NSGA II are convergence to the Pareto optimal front and diversity of solutions. Since the true Pareto front for this problem, like most real world problems, is not known, the first metric is not very useful. Another measure uses the spread of solutions along the optimal front. For any front, the spread of the solutions can be calculated by

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \bar{d} \right|}{d_f + d_l + \bar{d}(N-2)} \qquad 4.1$$

where d_f and d_l are the distances between extreme solutions and the boundary solutions obtained, d_i is the distance between two successive solutions, $\bar{d}$ is the average of all distances and N is the number of non-dominated solutions. The extreme solutions are found by fitting a curve parallel to the optimal Pareto front. For a set of solutions that uniformly cover the Pareto front, the value of Δ should be zero. If all the solutions lie at one point, Δ would be equal to one [22]. Due to the physical problem being solved, this is what is expected. Rather than directly calculating this parameter, the range and standard deviation of the solutions were examined; if the these values were small (on the order of 10^{-12}), the Pareto front was assumed to be a point.

The NSGA II program was very easy to apply. The user needs to specify only the objective functions, constraints, variable bounds, and a number of parameters. The function to calculate the objective values and constraints is the most difficult portion since it is the place where errors are most likely to arise. The user must set the upper and lower bounds for each variable. These are strictly enforced making implementing the endpoint constraints easy, because the bounds were just set to the variable's desired value.

The other parameters concern the functioning of the EA itself. The user must specify the population size and the number of generations the EA runs as well as the seed value for the random number generator. The probability of mutation and crossover occurring must also be set as well as the index of crossover and the index of mutation. The probability governs how often the corresponding operator is applied to the solution vector. The indices specify how far the offspring solutions are from the parents for the particular operator. A larger value for the index will produce offspring solutions closer to the parents, while for a smaller value, the offspring are farther away.

CHAPTER 5

RESULTS

5.1 Optimization Problem

Due to the nature of the EA used for optimization, the NLP problem derived in Chapter 3 is modified to make implementation easier. The objectives to minimize are still the defect value and time of flight, τ. However, just the maximum defect value is used as an objective instead of the vector of all defects, *d*, simplifying the problem immensely. The number of objects is only two instead of $6(N-1)$. Thus, the optimization problem transforms into

$$J(\mathbf{X}) = \begin{pmatrix} \max(d(\mathbf{X})) \\ \tau \end{pmatrix} \tag{5.1}$$

where $J(\mathbf{X})$, the set of objectives, subject to the equality constraints

$$\begin{gathered} \max\left(d(\mathbf{X})\right) = 0 \\ \rho_i = R_E \quad \omega_i = \omega_E \quad z_i = 0 \\ \rho_f = R_M \quad \omega_f = \omega_M \quad z = z_M \end{gathered} \tag{5.2}$$

and to the inequality constraints

$$\begin{aligned} \rho_i &\geq \rho_{i+1} \\ \theta_i &\leq \theta_{i+1} \\ z_i &\leq z_{i+1} \end{aligned} \tag{5.3}$$

Several formulations of position inequality constraints were tested, but this set consistently performed best.

5.2 Methodology

Several assumptions are made when implementing the trajectory optimization. The first simplification is assuming the orbits of Mercury and Earth to be circular with a radius equal to the semimajor axis of the planet's orbit. This allows the spacecraft to match the speed of Mercury without having to calculate its orbital position upon arrival since its orbit is fairly eccentric. The orbital parameters used are shown in Table 5-1 [23]. The Sun is assumed to be a point source of radiation with a constant luminosity of 3.827×10^{26} W ignoring the effects of its angular size and limb darkening. When the spacecraft reaches it, Mercury is assumed to be at its highest point above the Elliptic plane.

Table 5-1: Orbital Parameters for Earth and Mercury [23]

	Radius (AU)	Velocity (km/s)	Inclination
Mercury	0.387	48.87	7°
Earth	1.00	29.78	0°

For the solar sail spacecraft itself, the total mass is taken to be 500 kg. This includes the sail, supports, and payload. The sail itself is assumed to be square with a coefficient of reflection of 0.92, which is a reasonable value for an aluminum reflector. Factors such as deformation of the sail are also ignored. The sail control angles, α and δ, are allowed to vary between $-\pi/2$ and $+\pi/2$. At either extreme, the sail is edge-on to the incident solar radiation and receives no acceleration from it.

A canonical system of units is used for calculations. It is defined so that the gravitational parameter of the sun, μ_S, is equal to one and the distance unit is astronomical units where 1 AU = 1.496×10^{8} km. Thus, the canonical time unit is equal to 5.02276×10^{6} s or 58.13 days. In this system of units, the value of force due to the Sun's gravity at the Earth is 1 kg·AU/TU^2. The

magnitude of the acceleration experienced by a sail normal to the incident radiation at 1AU from the Sun is listed in Table 5-2 for each sail size.

Once the NSGA II program was confirmed to be working, a series of trails was run to find the best settings for the EA's parameters. The test was a 2D trajectory with 30 nodes for a 50 m sail. The initial generation was seeded with the possible ρ and θ solutions as described below and the program was run for 50,000 generations. For each series of trails, the parameter studied was allowed to vary while the others were held constant. The settings that correspond to the lowest defect values are used in the later trials.

To help speed convergence, the initial generation is seeded with several different ρ, θ and z configurations of possible solutions, as shown in Fig. 5-1. The first posits a linear relationship between the value and the node number. As the spacecraft gets closer to the planet, the node number increases. A half sinusoid is presumed in seeding the next quarter of the population. The third presumes a 1/n (where n is the current node) relationship with the variable and node. The last quarter of the population is still generated completely randomly to help maintain diversity. These non-random initial generations on average had a maximum defect of about 10^5 while for the random population members, it was on the order of 10^{11}.

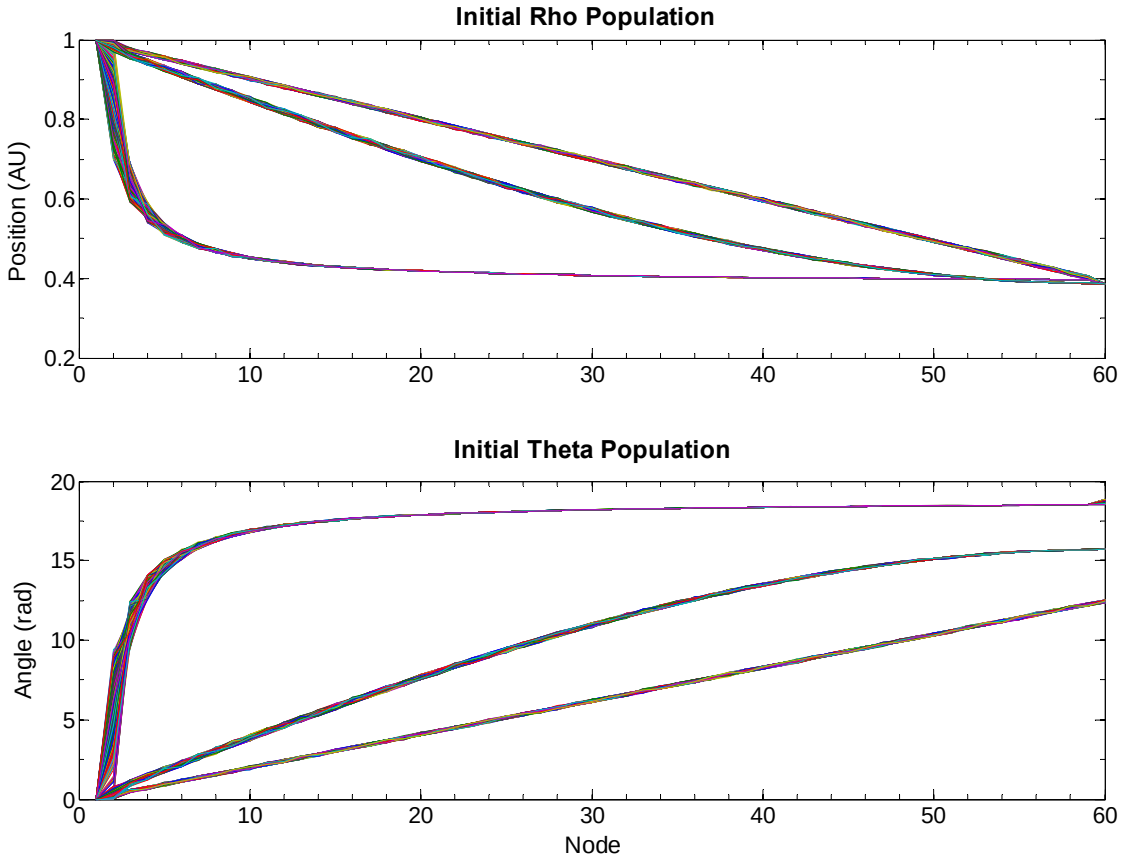

Figure 5-1: Non-random, initial generation ρ and θ components.

Several experiments were conducted to see the performance of NSGA II on this trajectory optimization problem. The first set tests the operation of the EA with several different formulations of the position constraints. Next, the performance of different sized sails, listed in Table 5-2, for a 2D trajectory was examined. This set of test cases was then repeated for a 3D trajectory.

Table 5-2: Side length, Area and σ values along with the acceleration experienced by a sail at 1 AU normal to the incident solar radiation.

Side Length (m)	Area (m^2)	σ (kg/m^2)	Acceleration (AU/TU^2)
50	2500	0.2	0.0147
100	10000	0.05	0.0588
125	15625	0.032	0.0918
150	22500	0.02222	0.1322
175	30625	0.01633	0.1800
200	40000	0.01250	0.2351

225	50625	0.00988	0.2975
250	62500	0.00800	0.3673
275	75625	0.00661	0.4444
300	90000	0.00556	0.5290
325	105625	0.00473	0.6208
350	122500	0.00408	0.7200
400	160000	0.00313	0.9404
450	202500	0.00247	1.1902
500	250000	0.002	1.4693

5.3 Results

The results of the numerical experiments are given below. In all cases, unless noted, all solutions lie along a sub-optimal Pareto front. The size of the front is a point with the average spread of solutions less than 10^{-12}. The solutions meet all constriants execpt the defect constraint. All plots give the average value of the variable being plotted. There is some variance in the variable vectors for each solution. It is the least for the θ component, and greatest for the control angles. However, it is small enough that the average is essentially the same.

5.3.1 Constraint Tests

Performance of NSGA II with five different formulations for the position inequality constraints are examined for their effect on the optimization. One trial was run for each case out to 300,000 generations on a 2D trajectory with 30 nodes. The first case places no constraints on position except at the endpoints. For the second case, the average value of a component over the trajectory is constrained to lie between the two endpoints.

$$\begin{aligned} \rho_i \geq \overline{\rho} \geq \rho_f \\ \theta_i \leq \overline{\theta} \leq \theta_f \\ z_i \leq \overline{z} \leq z_f \end{aligned} \qquad 5.4$$

The third case uses the trend of the previous ten nodes. The value of a component at a particular node had to be less (greater for θ and z components) than the average value over the previous ten nodes.

$$\begin{aligned} \frac{1}{10}\left(\sum_{k=0}^{9}\rho_{i-k}\right) &\geq \rho_{i+1} \\ \frac{1}{10}\left(\sum_{k=0}^{9}\theta_{i-k}\right) &\leq \theta_{i+1} \\ \frac{1}{10}\left(\sum_{k=0}^{9}z_{i-k}\right) &\leq z_{i+1} \end{aligned} \qquad 5.5$$

The next case is similar except that the trend is only over the previous five nodes.

$$\begin{aligned} \frac{1}{5}\left(\sum_{k=0}^{4}\rho_{i-k}\right) &\geq \rho_{i+1} \\ \frac{1}{5}\left(\sum_{k=0}^{4}\theta_{i-k}\right) &\leq \theta_{i+1} \\ \frac{1}{5}\left(\sum_{k=0}^{4}z_{i-k}\right) &\leq z_{i+1} \end{aligned} \qquad 5.6$$

The final case employs the constraints given in equation 5.3 where each value at a node had to be less or greater than at the previous one. The results of the trials are shown in Table 5-3 and Figure 5-2.

Table 5-3: Defects and times obtained from constraint tests

	Defect	Time (yrs)
No Constraints	0.202668	2.37861371
Average	4.780038	7.12062368
10 Node Trend	0.358292	2.31430921
5 Node Trend	1.695938	10.9888090
All Constraints	0.289080	1.73369862

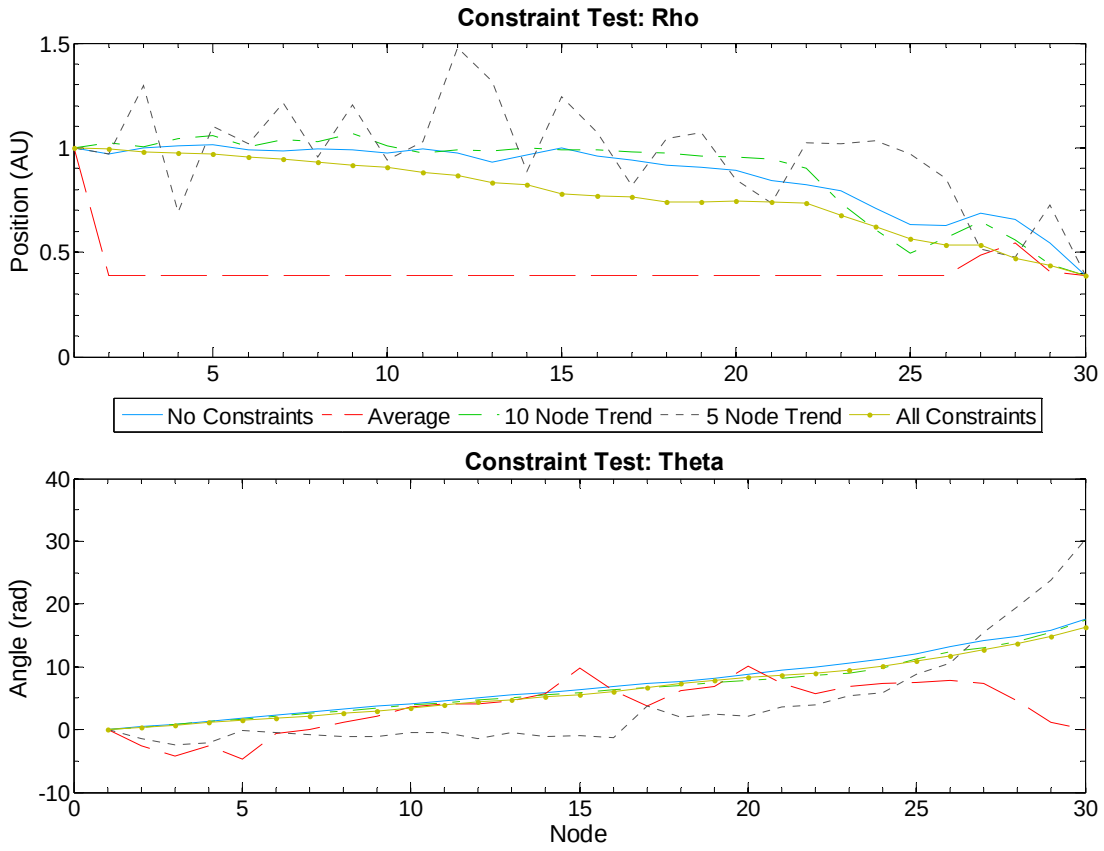

Figure 5-2: Average values of ρ and θ components at each node for the constraints tested.

The equality constraints applied to the position variables definitely affects the performance of the optimization algorithm. The case that used the average value over the trajectory preformed the worst in terms of the defect and the trajectory produced. It is unrealistic because the trajectory quickly gets to the orbit of Mercury and oscillates around $\theta = 0$. The case using the trend of the previous 5 nodes was the worst in terms of time, taking over 10 years to get to Mercury. The other three cases produced similar results. Surprisingly, the case with no constraints obtained a lower defect value than the case with all constraints. However, when run on a more complex case with 60 nodes, it performs much worse. The ρ component looks unphysically jagged and the time and defect are higher than with all the constraints in place. It appears that when NSGA II is allowed to run for a sufficiently large number of generations, the

trajectory will begin to smooth out even without the position constraints in place since it is more optimal. However, on more difficult problems the inequality constraints placed on the position components of the variable vector accelerate this process. It also prevents the spacecraft from staying in orbit around the Earth with the sail edge-on to the incident radiation.

5.3.2 Two Dimensional Trajectory

This next experiment studies the performance of different sized sails for 2D trajectories. First, NSGA II was run to one million generations for each sail. The results are shown in Figure 5-3. The defects from these runs are fairly widely varied, but there is a general trend of the times as wells as the defect increasing with sail size. To see if this is an artifact of the optimizer, an attempt was made to obtain the same defect value for each sail area. These runs were seeded with the average solution vector for each sail from the previous run and half completely random population members. Defect and time results are shown in Figure 5-4.

For sails with sides between 50 m and 325 m, the defect value reached was 0.1975 ± 0.0003. The time of flight ranges from 2.0719 yrs to 2.21 yrs and tends to increase for larger sail areas. Reaching this defect value was attempted for all sizes, but for sails with side lengths larger than 325 m, it was not possible in the given time. After running NSGA II for an additional two to

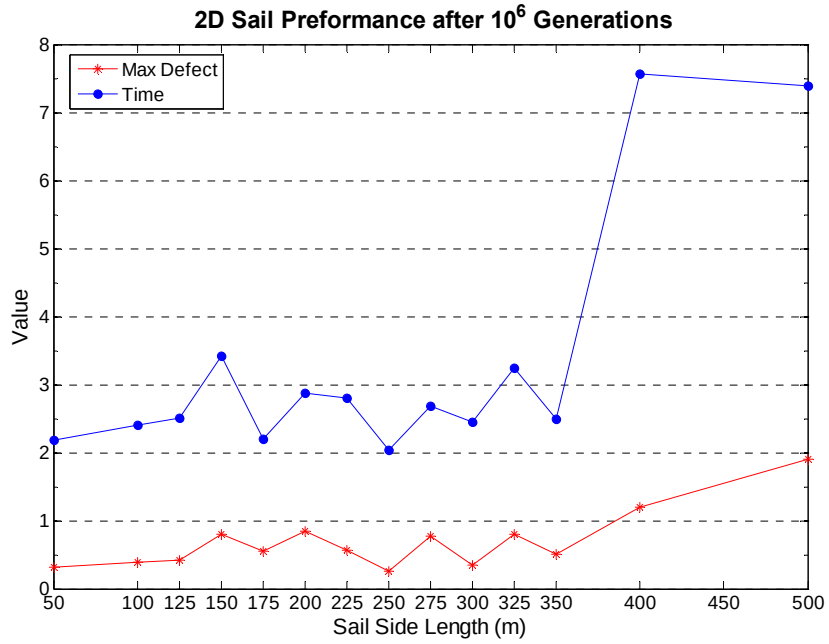

Figure 5-3: Results for different sized sails for 2D trajectory after 10^7 generations.

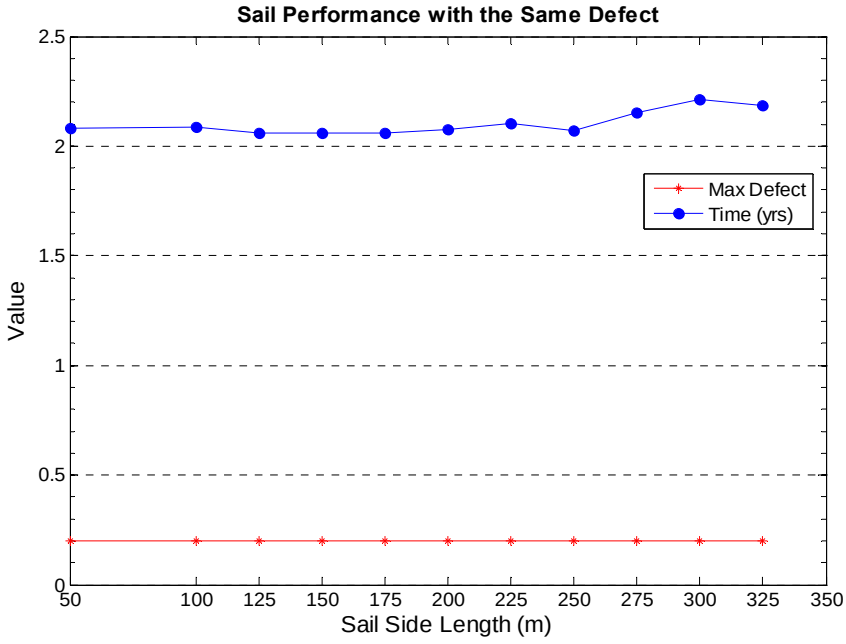

Figure 5-4: Results for different sized sails with the same defect value.

3×10^7 generations in some cases, the attempt was halted. The results for these sail areas is in Table 5-4. For these sails, the acceleration from solar radiation pressure is at least half of the gravitational acceleration at the Earth for a sail face-on to the Sun and would increase as it spirals toward Mercury. This could cause some control problems with the optimizer. It could also be that the resolution is too low for these cases, and to drive the defects down any further more nodes would have to be added.

Table 5-4: Final results for larger sails

Side Length (m)	Defect	Time (yrs)
350	0.3749	2.5089
400	0.2718	2.3233
450	0.3929	2.5634
500	0.3862	2.5631

For the smaller sails, several variables and the trajectories for selected sail sizes are plotted in Figures 5-5 to 5-8. The sails with side lengths between 50 m and 175 m had nearly identical performance. Although the larger sail does have a slight advantage with a time of flight shorter by nearly 12 days. To see if a 175 m sail functions significantly better than the 50 m sail, a more optimal solution is needed.

The sail with a side length of 275 m is the most unusual. It arrives at the position of Mercury in the fewest number of nodes and orbits with the planet. If it reaches the planet that quickly, it is surprising the time required is long compared to the others. This sort of behavior is seen in the trajectories of sails larger than 350 m.

Looking at the trajectories plotted in Figure 5-7 and Figure 5-8, as the spacecraft first leaves Earth's orbit, larger sails tend to lead the small ones. However, after about one and a half revolutions this trend does not hold for the 275 m and 50 m sails. For the small sails which spend less time edge to the incident solar radiation, they could gain more inward acceleration than a larger sail where the outward radiation pressure may be a problem.

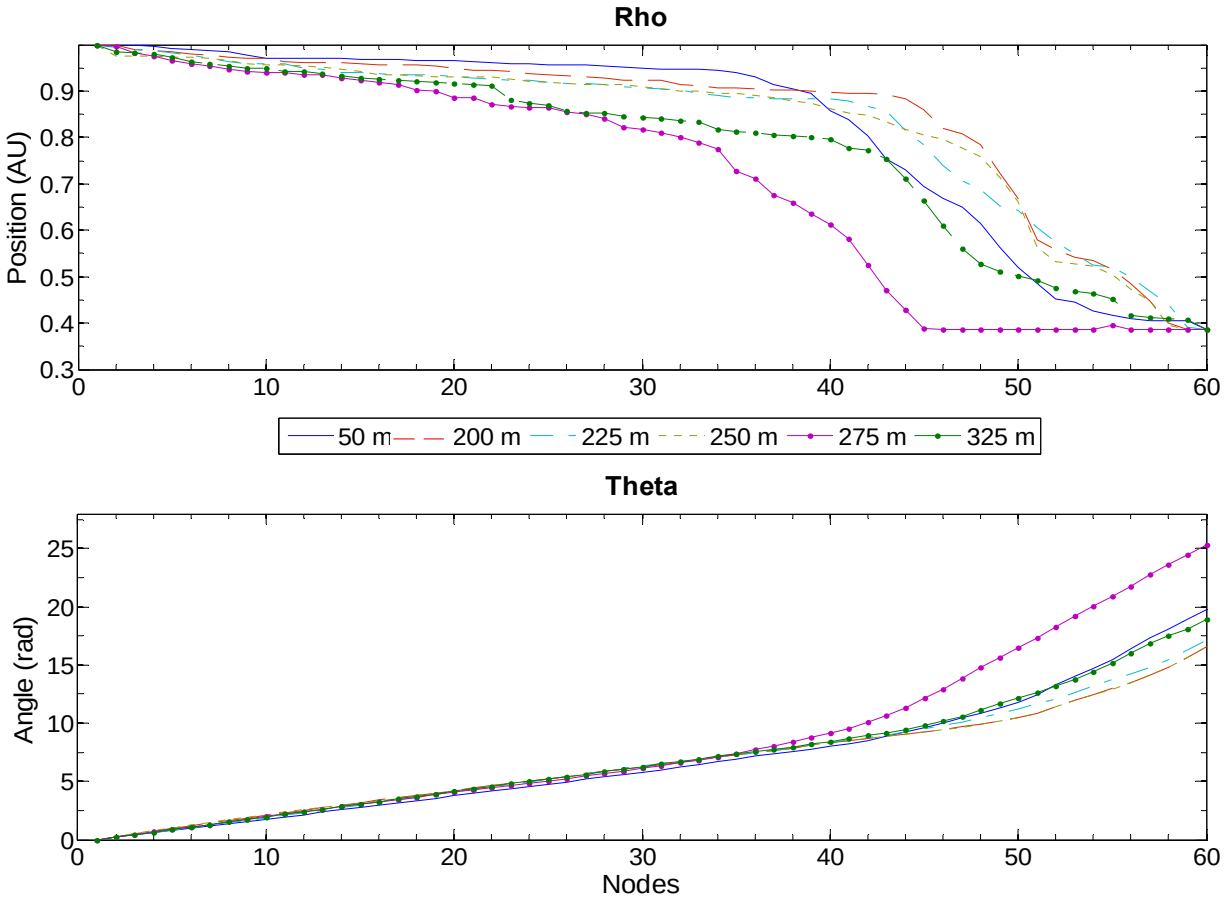

Figure 5-5: ρ and θ for selected sail areas with the same defect value.

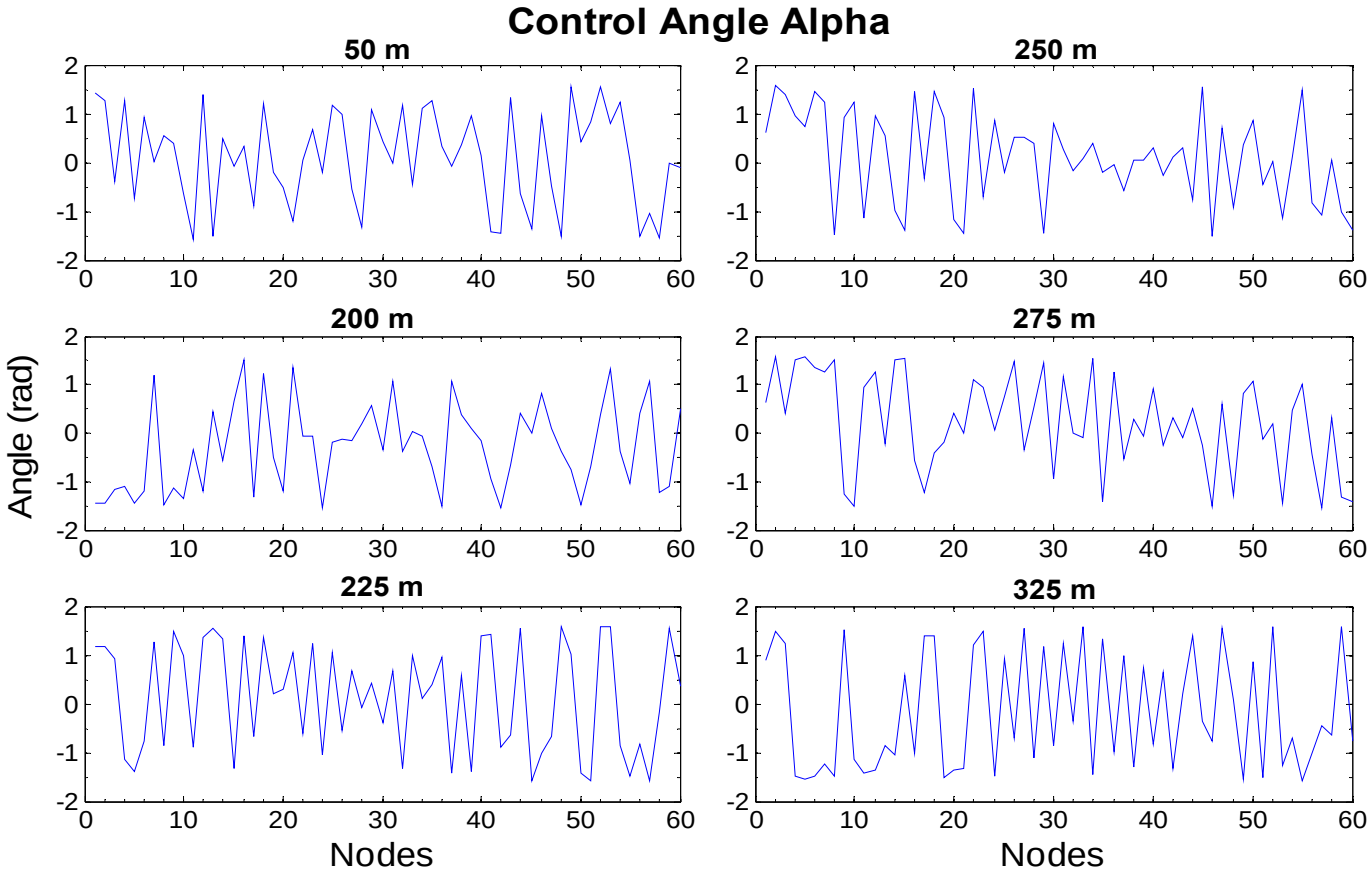

Figure 5-6: Corresponding control angles

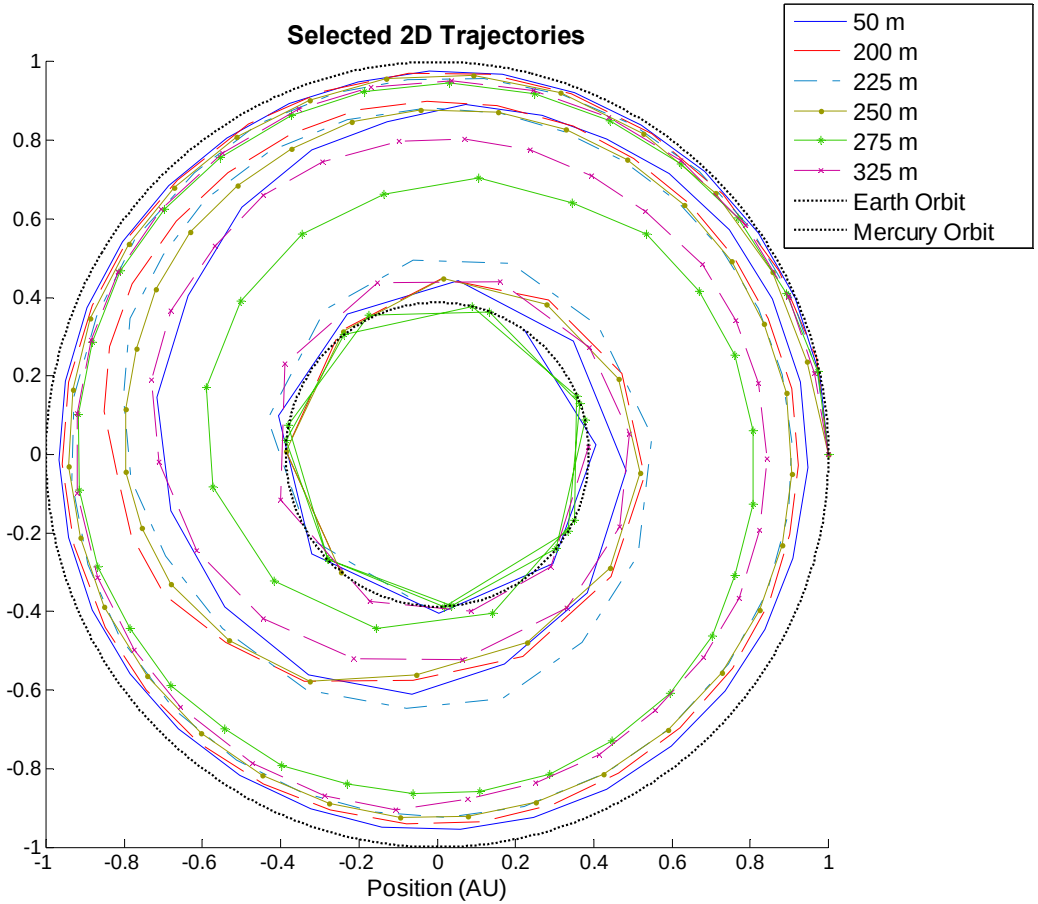

Figure 5-7: 2D trajectories for the selected sail sizes

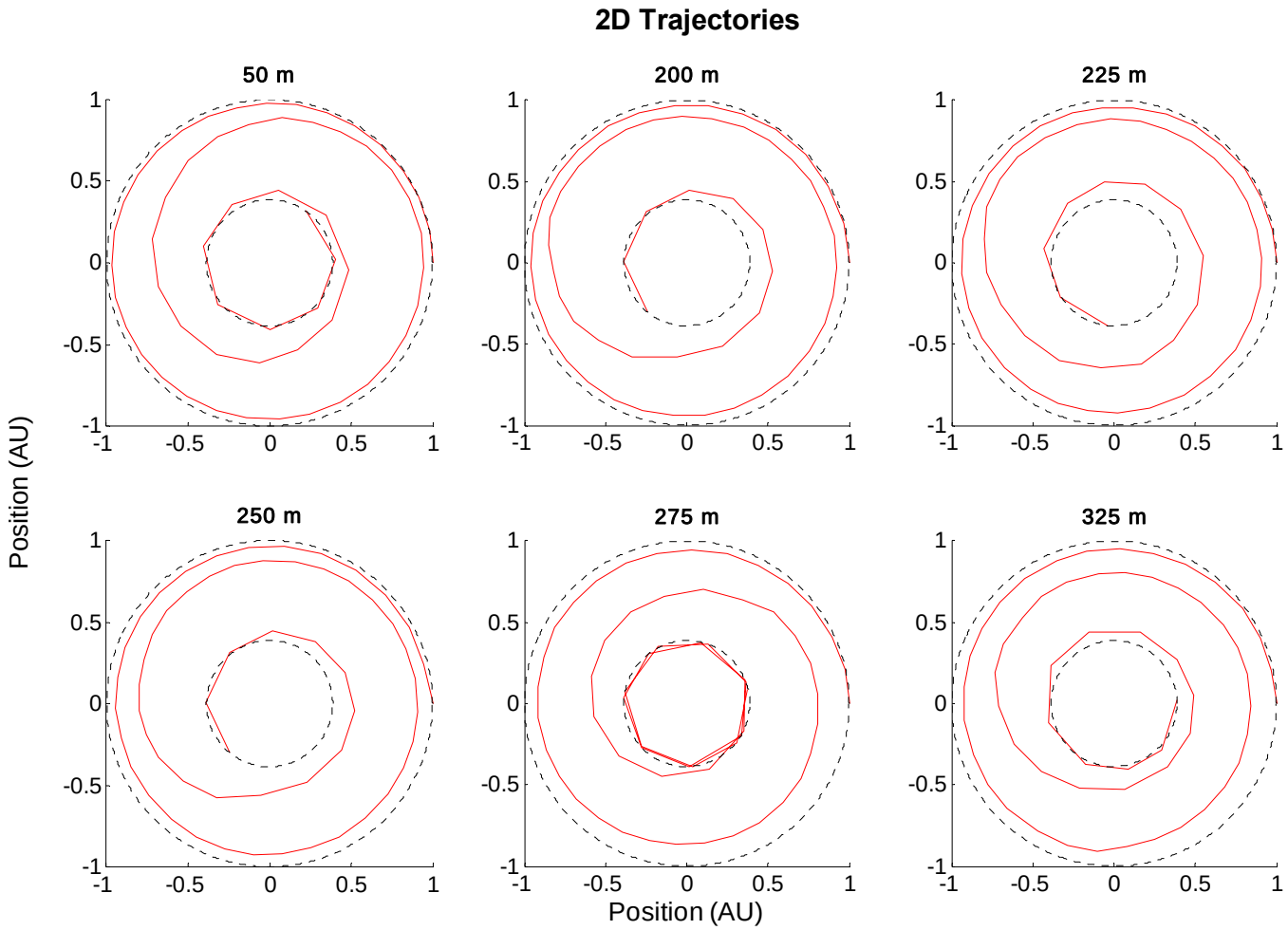

Figure 5-8: 2D Trajectories for selected sail sizes plotted separately. The dashed lines are the orbits of Earth and Mercury.

5.3.3 Three Dimensional Trajectory

The final set of tests is similar to the previous one, except the performances of various sail areas are tested for a three dimensional trajectory. The number of sail sizes examined was reduced because the execution times to get a low defect values were much longer than the 2D case. Although the actual value of z for Mercury is only 0.075AU, the addition of the third dimension to the problem greatly increases the difficulty of the optimization. The space that NSGA II has to search is larger and the number of variables actually solved for in the state vector has increased from $6N+1$ to $8N+1$. Thus, the program requires many more generations for acceptable convergence. An initial case with a 2500 m^2 sail was run for 2.2×10^7 generations resulting in a defect of 1.37. The average state vector corresponding to this solution was then used to seed half the population members for other sails. The trial for each was run for one million generations. The results are shown in Figure 5-9. An attempt was made to drive down the defects to the same value as in the previous 2D experiment. They were halted after running for 3×10^7 additional generations with very little improvement in the defect value. It could be that more nodes would permit a more accurate solution for the 3D trajectory optimization problem. The results for all cases are shown in Figures 5-9 to 5-14.

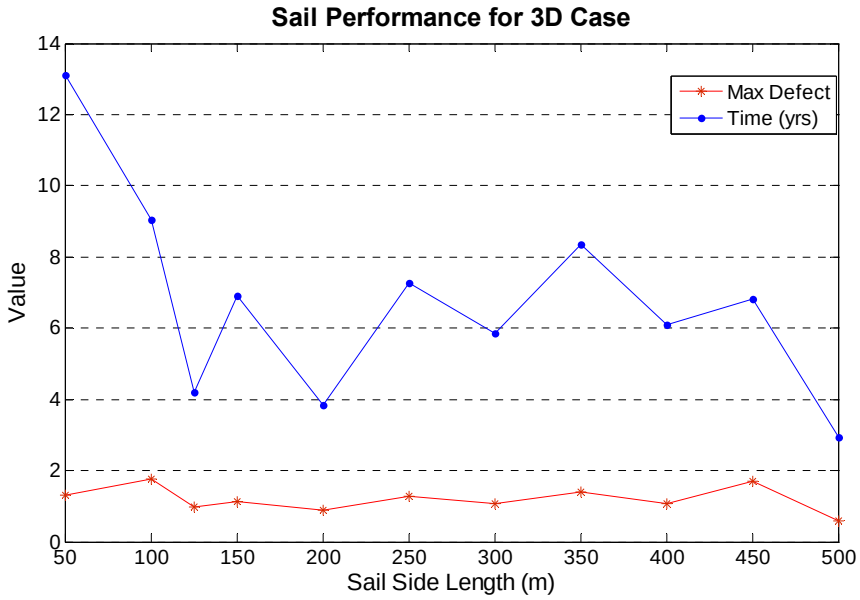

Figure 5-9: Results of objective values for 3D trajectories for various sail sizes.

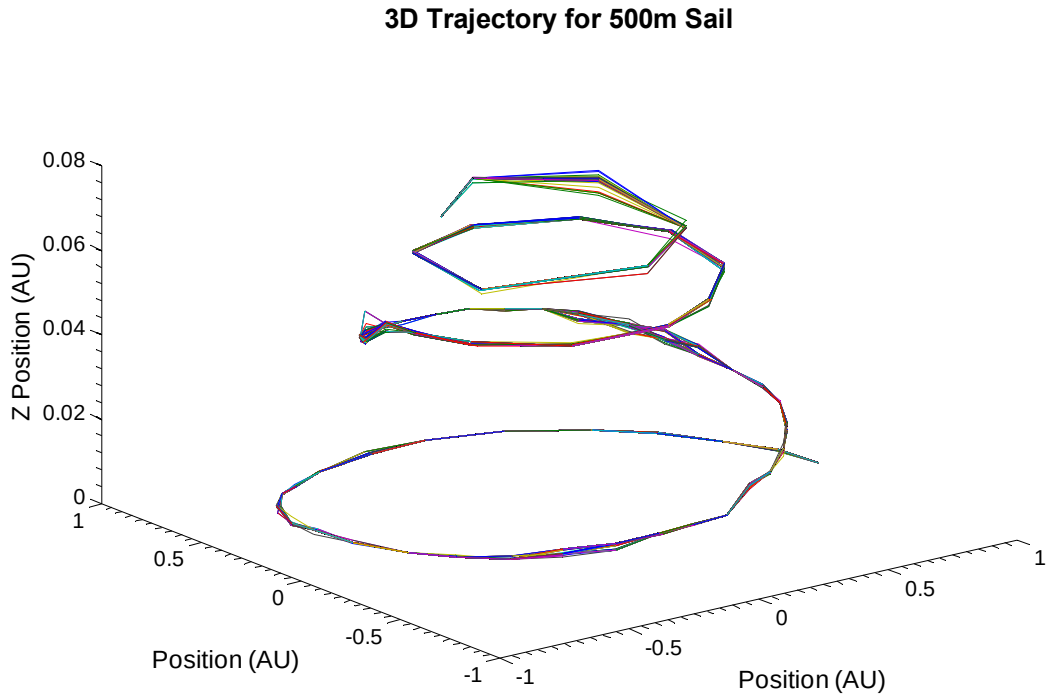

Figure 5-10: Trajectory of a sail with side-length of 500 m corresponding to the best solution found for the 3D case. All population members are plotted.

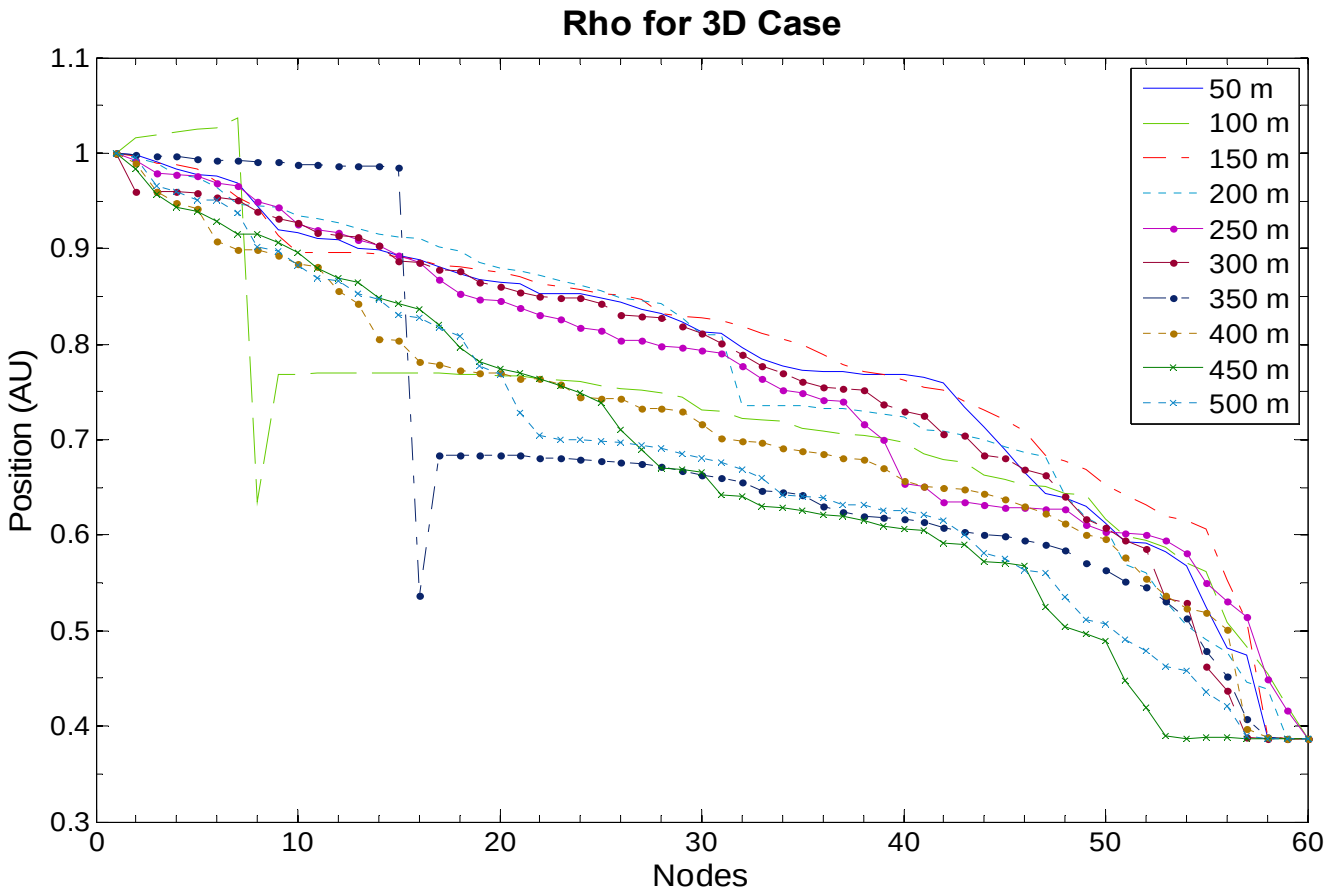

Figure 5-11: Plot of the ρ component for sails in the 3D case.

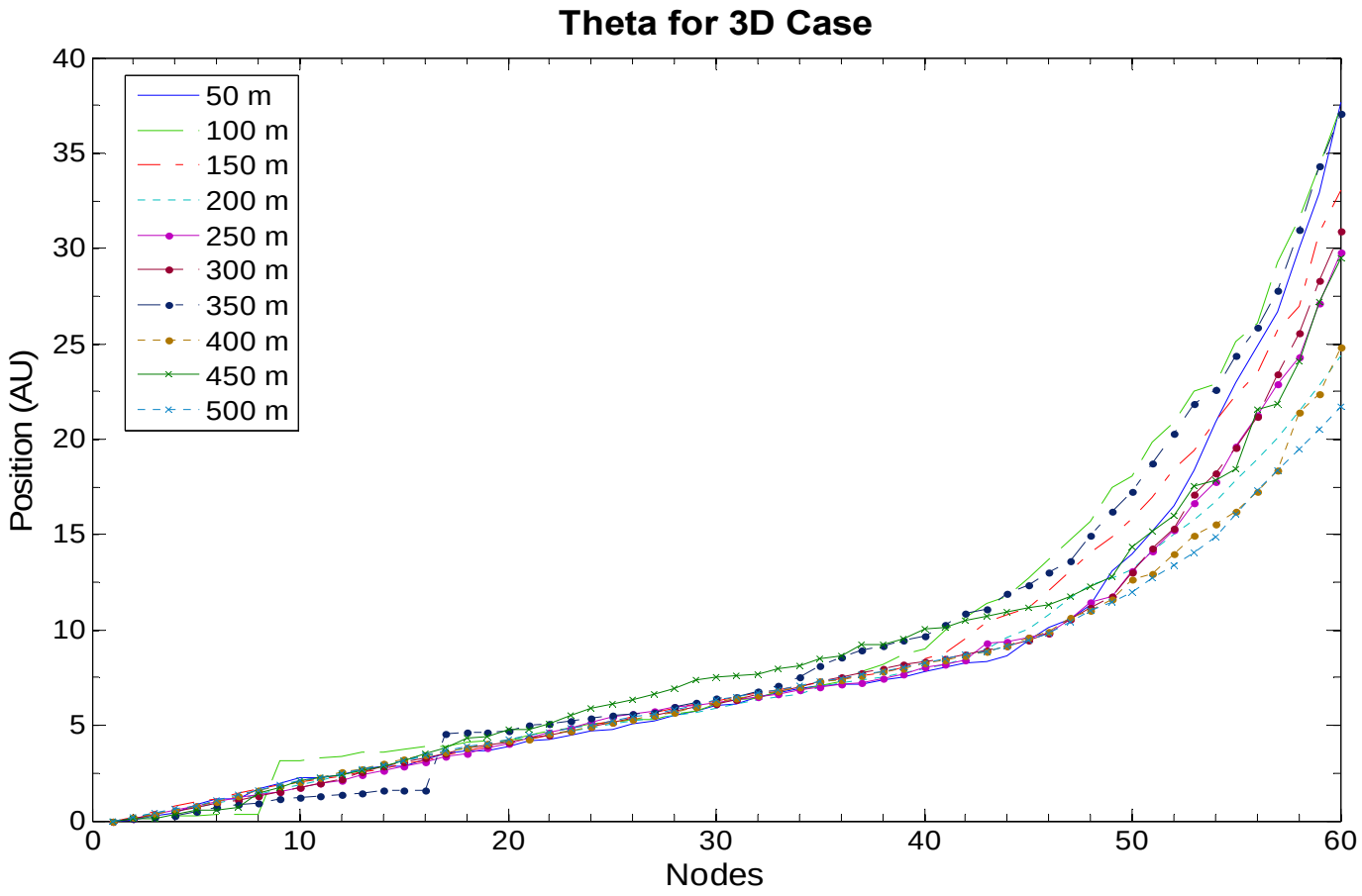

Figure 5-12: Plot of the θ component for sails in the 3D case.

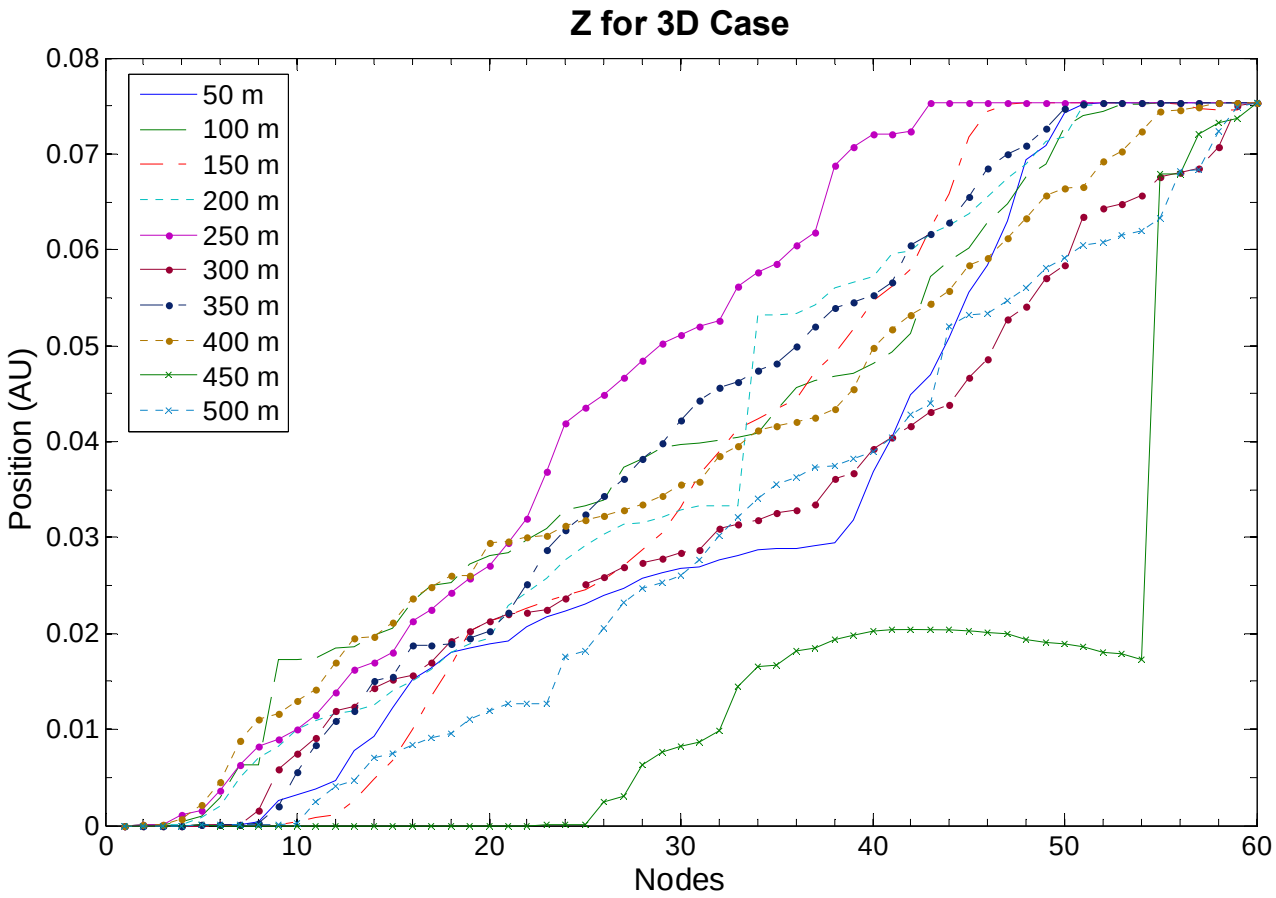

Figure 5-13: Plot of the z component for sails in the 3D case.

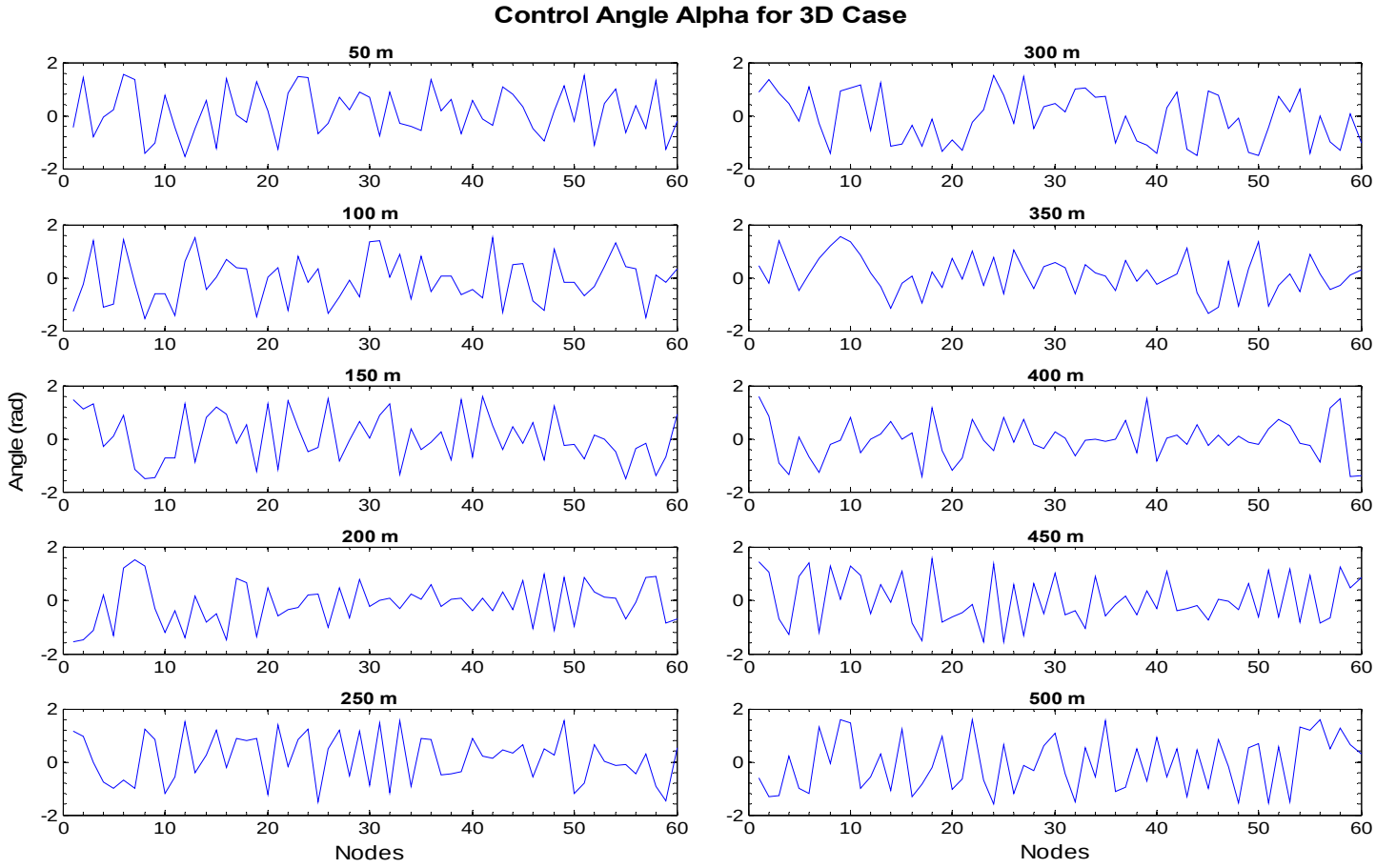

Figure 5-14: Time history of the in-plane control angle α for each sail.

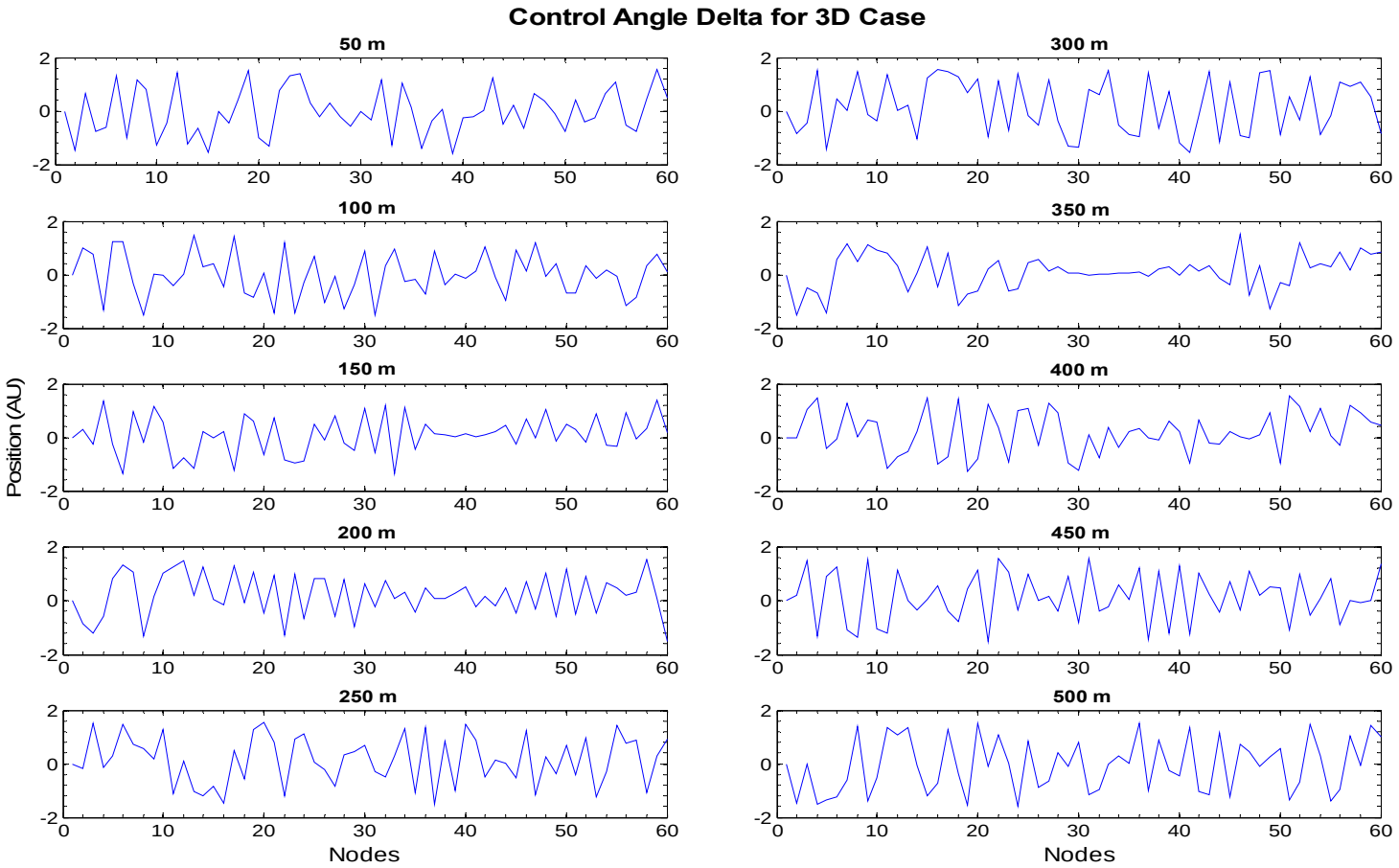

Figure 5-15: Time history of the out-of-plane control angle δ for each sail.

For the 3D case, the values of the maximum defects are larger than the corresponding 2D trajectories and range from 1.76 for a 100 m sail to 0.576 for a 500 m sail which also has the shortest time of 2.92 yrs. No general trend can be seen between time of flight, defect, and sail size. There are large discontinues in the ρ and z components due to the poorness of the approximation. Interestingly, mid-sized sails seem to spend more time face-on to the Sun than the largest and smallest sails.

CHAPTER 6

SUMMARY AND CONCLUSIONS

Using the method of direct collocation with an evolutionary algorithm, NSGA II, sub-optimal trajectories from Earth to Mercury for a spacecraft using solar sail propulsion are found for both 2D and 3D trajectories. For the 2D case, it was much easier to find solutions with values for the maximum defect less than one. Larger sails tended to have corresponding larger defects than smaller sails. The times-of-flight for the spacecraft are all between two and three years. These are longer than 0.5 yr to 1.5 yrs flight times given in the literature for various sail sizes [12], but not by much. The solutions for the 3D trajectories did not converge to as low a maximum defect value for a sail of the same size. It also took many more generations to do generate an acceptable solution. In part this was expected, because of the increase in problem size, but the number of additional generations required was surprising.

The next logical step for further research would be to feed the results from NSGA II into a standard NLP solver. Comparing the solutions would give a better idea about how close the trajectories generated by NSGA II are to being truly globally optimal. Another useful study would be to increase the number of nodes to see if the defects decrease further, especially in cases with large defect values. The current solutions could be interpolated and then used as seeds in the initial generation. However, depending on the number of nodes added, this could be computationally expensive.

With the formulation for the acceleration derived in terms of an intrinsic stellar property, luminosity, the possibility of using solar sailing around another star arises. Depending on the type of star, different materials might be needed for the reflecting surface, such as silver. Technologically this may never be feasible, but it is an interesting conjecture.

The combination of the methods used does work to generate sub-optimal solutions in a few hours to days, but there probably is a better approach. A simple single-objective EA focusing only on the defect could probably be used instead of the multi-objective NSGA II leaving time to be optimized when the solution is fed into a traditional NLP solver.

BIBLIOGRAPHY

1) Enright, P. J., and Conway, B. A., "Optimal Finite-Thrust Spacecraft Trajectories Using Collocation and Nonlinear Programming," *Journal of Guidance, Control and Dynamics*, Vol. 14, No. 5, 1991, pp.981-985.

2) Tang, S., and Conway, B.A., "Optimization of Low-Thrust Interplanetary Trajectories Using Collocation and Nonlinear Programming," *Journal of Guidance, Control, and Dynamics,* Vol. 18, No. 3, 1995, pp. 599-604.

3) Scheel, W.A., and Conway, B.A., "Optimization of Very Low-Thrust, Many-Revolution Spacecraft Trajectories," *Journal of Guidance, Control, and Dynamics,* Vol. 17, No. 6, 1994, pp. 1185-1192.

4) McInnes, C. R., Hughes, G., and McDonald, M., Low cost Mercury orbiter and sample return missions using solar sail propulsion," The Aeronautical Journal, Vol. 107, No. 1074, 2003, pp. 469-478.

5) Coverstone-Carroll, V., Hartmann, J.W, Mason, W. J., "Optimal Multi-Objective Low-Thrust Spacecraft Trajectories," *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, 2000, pp. 387-402.

6) Yokoyama, N., and Suzuki, S., "Modified Genetic Algorithm for Constrained Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol 28, No. 1, 2005, pp.139-44.

7) Wall, B., and Conway, B.A, "Near-Optimal Low-Thrust Earth-Mars Trajectories via a Genetic Algorithm," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, pp. 1027-1031, 2005.

8) Lee, S., von Allmen, P., Fink, W., Petropoulos, A. E., and Terrile, R. J., "Comparison of Multi-Objective Genetic Algorithms in Optimizing Q-Law Low-Thrust Orbit Transfers," *Genetic and Evolutionary Computation Conference 2005,* Washington, DC, 2005.

9) Vasile, M., Summerer, L., De Pascale, P., "Design of Earth-Mars Transfer Trajectories Using Evolutionary –Branching Technique," *Acta Astronautica*, Vol. 56, 2005, pp. 705-720.

10) Dachwald, B., "Optimization of very-low-thrust trajectories using evolutionary neurocontrol", *Acta Astronautica,* Vol. 57, 2005, pp. 175-185.

11) Radice, G. and Olmo, G. "Ant Colony Algorithms for Two-Impulse Interplanetary Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, 2006, pp.1140-1143.

12) McInnes, C. R., *Solar Sailing: Technology, Dynamics, and Mission Applications*, Springer-Praxis, Chichester, 1999, pp. 1-150.

13) Koblik, V. Polyakhova, E., Sokolov, L., "Controlled Solar Sail Transfers into the Near-Sun Regions combined with Planetary Gravity-Assist Flybys," *Celestial Mechanics and Dynamical Astronomy,* Vol. 86, 2003, pp. 59-80.

14) Wright, J. L., *Space Sailing*, Gordon and Breach Science Publishers, Philadelphia, 1992.

15) Leipold, M.; Borg, E.; Lingner, S.; Pabsch, A.; Sachs, R.; Seboldt, W. "Mercury Orbiter with a Solar Sail Spacecraft," *Acta Astronautica*, Vol. 35, 1995, pp. 635-644.

16. Johnson, L., Young, R.M., and Montgomery, E. E., "Status of Solar Sail Propulsion: Moving Toward an Interstellar Probe", *New Trends in Astrodynamics and Applications III. AIP Conference Proceedings*, Vol. 886, 2007, pp. 207-214.

17. Friedman, L., "Solar Sail Update: The End of Cosmos 1; the Beginning of the Next Chapter," *The Planetary Society News*, 30 September 2005

18) Dachwald, B., "Optimal Solar-Sail Trajectories for Missions to the Outer Solar System," *Journal of Guidance, Control and Dynamics*, Vol. 28, No. 6, 2005, pp. 1187-1193.

19) Matloff, G., Taylor, T., Powell, C., Moton, T., "Phobos/Deimos Sample Return via Solar Sail" *Annals of the New York Academy of Science*, Vol. 1065, 2005 pp.429-440.

20) Betts, J.T, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193-207.

21) *NSGA II Code in C*, Kanpur Genetic Algorithms Laboratory, http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-v1.1.tar,

22) Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T., "A Fast and Elitist Multi-Objective Genetic Algorithm-NSGA-II," *KanGAL Report Number 2000001,* 2000.

23) Reed, P., "Class Notes", *Systems Optimization Using Evolutionary Algorithms,* Department of Civil Engineering, The Pennsylvania Sate University, Spring 2007.

24) Deb, K. and Kumar, A. "Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems," *Complex Systems*, Vol. 9, No. 2, 1995, pp. 431-454.

25) "Mercury Fact Sheet," *NASA Planetary Fact Sheets,* http://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html, 20 March 2008.

APPENDIX

OBJECTIVE CALCULATION CODE

The code used to calculate the objective values and constraints in the NSGA II program.

```
/* objective problem definitions */

# include <stdio.h>
# include <stdlib.h>
# include <math.h>

# include "global.h"
# include "rand.h"

// function prototypes
double* deriv(double state[6],double cntrl[2]);
double*     defectCalc(double     xLt[6],double     xRt[6],double
xLtDot[6],double xRtDot[6],double uLt[2],double uRt[2], double tSeg);

// main objective function ie calculates the defects and
//assigns them as objective values and assigns constraints
void  obj_Calc(double  *xreal,  double  *xbin,  int  **gene,  double
*obj, double *constr)
{

//initialize variables
double *xDot_Ptr;
double *currentDefect_Ptr;
double xDot[3];
double currentDefect[6];


//initialize arrays
double s[6]       =  {0.0,0.0,0.0,0.0,0.0,0.0};  //  state  vector
for a particular node
double sDot[6]    =  {0.0,0.0,0.0,0.0,0.0,0.0};  //  derivative  of
state vector for a particular node
double u[2]       = {0.0,0.0};                   // control vector
at a particular node


double  xL[6]     = {0.0,0.0,0.0,0.0,0.0,0.0}; // state vector at
left node
double  xR[6]     = {0.0,0.0,0.0,0.0,0.0,0.0}; // state vector at
right node
double  xLDot[6]  =  {0.0,0.0,0.0,0.0,0.0,0.0};  //  derivative  of
state vector at left node
double  xRDot[6]  =  {0.0,0.0,0.0,0.0,0.0,0.0};  //  derivative  of
state vector at right node
```

```
      double uL[2]    = {0.0,0.0};  // control vector at left node
      double uR[2]    = {0.0,0.0};  // control vector at right node

      double totalDefect[numDef];

      // counting variables
      int n = 0;
      int j = 0;
      int i = 0;
      int L = 0;
      int m = 0;
      int q = 0;
      int K = 0;
      int b,w;
      double max_defect=-1.0e12;

      // initialize time variables
      double totalTime = 0.0;  // total time of flight
      double tSeg = 0.0;       // time of flight between nodes

      // constraint variables
      double r_in,r_fin,t_in,t_fin,z_in,z_fin;
      double r_curr,r_sum,r_av,t_curr,t_sum,t_av;//,z_curr,z_sum,z_av;


      totalTime = xreal[numVar-1];  // total time is a variable to be
optimized
      tSeg = totalTime /(double)nodes;


      for (n=0;n < nodes; n++){ // at each node

      // set state and control vectors with variables at the node
                  s[0] =xreal[m];   // rho
                  s[1] =xreal[m+1]; // theta
                  s[2] =xreal[m+2]; // z
                  s[3] =xreal[m+3]; // rho dot
                  s[4] =xreal[m+4]; // theta dot
                  s[5] =xreal[m+5]; // z dot
                  u[0] = xreal[m+6]; // alpha
                  u[1] = xreal[m+7]; // delta

      // get derivatives
      xDot_Ptr = deriv(s,u);

      for (q=0;q<3;q++){
            xDot[q] = *xDot_Ptr++;
            }// end for

            // form state dot vector
                  sDot[0] =xreal[m+3];    // rho dot
                  sDot[1] =xreal[m+4];    // theta dot
                  sDot[2] =xreal[m+5];    // z dot
                  sDot[3] =xDot[0]; // rho double dot
                  sDot[4] =xDot[1]; // theta double dot
```

```
                  sDot[5] =xDot[2]; // z double dot

            // set state vector as xR
                  if (n >= 1){

                  // set xR
                  for(j = 0; j < 6; j++){
                        xR[j] = s[j];
                  // derivative of state vector at left most node
                        xRDot[j] = sDot[j];
                  //contols at left most node
                        if(j < 2){
                              uR[j] = u[j];
                        }// end if
                  }// end for

                  // calculate defect
                  currentDefect_Ptr = defectCalc(xL, xR, xLDot, xRDot,
uL, uR,tSeg);

                  for (q=0;q<6;q++){
            currentDefect[q] = *currentDefect_Ptr++;
            }// end for

                  //update xL

                  for(j = 0; j < 6; j++){
                        xL[j] = s[j];
                  // derivative of state vector at left most node
                        xLDot[j] = sDot[j];
                  //contols at left most node
                        if(j < 2){
                              uL[j] = u[j];
                        }// end if
                  }// end for

      // make array of total defecta using absolute value a defect

            totalDefect[K]     = fabs(currentDefect[0]);
            totalDefect[K+1] = fabs(currentDefect[1]);
            totalDefect[K+2] = fabs(currentDefect[2]);
            totalDefect[K+3] = fabs(currentDefect[3]);
            totalDefect[K+4] = fabs(currentDefect[4]);
            totalDefect[K+5] = fabs(currentDefect[5]);
            //totalDefect[m+6] = currentDefect[6];
            //totalDefect[m+7] = currentDefect[7];

            K +=6;

            }// end if
            else{//for the first state vector, set it equal to xL
                  // state vector at left most node
                              for(j = 0; j < 6; j++){
                                    xL[j] = s[j];
```

```
                              //  derivative  of  state  vector  at  left
most node
                                    xLDot[j] = sDot[j];
                              //contols at left most node
                                    if(j < 2){
                                          uL[j] = u[j];
                                    }// end if
                              }// end for

            }//end else

      m +=8;
      }//end for each node

      // assign defects as objective values and constrain to be
      // >=1e-14

      /* //for using all defects as objectives
      b =0;
      for (i=0;i<numDef; i++){
                  obj[b]= totalDefect[i];
                  constr[b] = 1.0e-6 - totalDefect[i];

                  if (totalDefect[i] > max_defect){
                        max_defect = totalDefect[i];
                  }
                  b++;
            }// end for*/


       //for just max defect
      for (i=0;i<numDef; i++){
            if (totalDefect[i]> max_defect){
                  max_defect = totalDefect[i];
            }

            }// end for

            obj[0]= max_defect; // defect objective
      constr[0] = 1.0e-14 -max_defect;// defect Constraint
            b = 1;


      // Position Constraints

      //each value greater or smaller than value at previous node
      for (w =1; w<nodes; w++){

                        j= w-1;

                        r_in = xreal[j*8];
                        r_fin  = xreal[w*8];

                        t_in = xreal[(8*j)+1];
                        t_fin  = xreal[(8*w)+1];
```

```
                        z_in = xreal[(8*j)+2];
                        z_fin  = xreal[(8*w)+2];

                        constr[b] = r_in - r_fin;
                        b++;
                        constr[b] = t_fin - t_in;
                        b++;
                        constr[b] = z_fin - z_in;
                        b++;
      }// end for

      /*  //  use  trend  over  whole  trajectory  -  average  rho/theta
small/larger //than endpoints

            j= 8*(nodes-1); //index of last node position
            r_in = xreal[0];
            t_in = xreal[1];

            r_in = xreal[j];
            t_in = xreal[j+1];

            r_sum =0;
            t_sum =0;

      for (w =0; w<nodes; w++){
                  r_sum += xreal[w*8];
                  r_sum += xreal[w*8];

      }// end for

            r_av = r_sum/(double)nodes;
            t_av = t_sum/(double)nodes;

            constr[1] = r_in - r_av;
            constr[2] = r_av - r_fin;

            constr[3] = t_av - t_in;
            constr[4] = t_fin - t_av;
      */
      /*//trend over previous 5 nodes so rho/theta small/larger
            for (w =4; w<nodes; w++){

                  r_sum =0;
                  t_sum =0;
                  //z_sum =0;

                  r_curr =xreal[w*8];
                  t_curr =xreal[w*8+1];
                  //z_curr =xreal[w*8+2];

                  for(L=0;L<10;L++){
                        r_sum += xreal[(w-L)*8];
                        t_sum += xreal[((w-L)*8)+1];
                        //z_sum += xreal[((w-L)*8)+2];
```

```
                  }//end for
                        r_av = r_sum/5.0;
                        t_av = t_sum/5.0;
                        //z_av = z_sum/5.0;

                        constr[b] = r_av -r_curr;
                        b++;
                        constr[b] = t_curr - t_av;
                        b++;
                        //constr[b] = z_curr - z_av;
                        //b++;

      }// end for
      */


      //obj[numDef] = totalTime; //set time as final objective when
using //all defects
      obj[1] = totalTime;

      return;

      }//end main function


      // sub functions

      /*****************************************************/
      double *deriv(double state[6],double cntrl[2]){

            static double accel[3] ={0.0,0.0,0.0};
            double *accel_ptr;
            //int k;

            double rho,theta,ze,rhoD,thetaD,zD;

      //declare and initialize convenience variables
            double R = 0.0;
            double r2     = 0.0;
            double r3     = 0.0;
            double cDcA   = 0.0;
            double cDsA   = 0.0;
            double sD     = 0.0;
            double rR     = 0.0;
            double zR     = 0.0;
            double lambda = 0.0;

            double rhoDD    = 0.0;
            double thetaDD  = 0.0;
            double zDD        = 0.0;

      // rename vector components
            rho         = state[0];
            theta = state[1];
```

```
            ze          = state[2];
            rhoD  = state[3];
            thetaD      = state[4];
            zD          = state[5];

            // calculate convenience variables
             R = sqrt(pow(rho,2.0) + pow(ze,2.0));
             r2 = pow(R,2);
             r3 = pow(R,3);
             cDcA = cos(cntrl[1])*cos(cntrl[0]);
             cDsA = cos(cntrl[1])*sin(cntrl[0]);
             sD = sin(cntrl[1]);
             rR = rho/R;
             zR = ze/R;
             lambda = pow((rR*cDcA + zR*sD),2.0);


      // calculate derivatives
            //rho acceleration
            rhoDD    =    (OM/r2)*lambda*cDcA    -    (MU*rho)/r3    +
pow(thetaD,2)*rho;
            //theta acceleration
            thetaDD   =   (1/rho)*((OM/r2)   *   lambda   *   cDsA   -
2*rhoD*thetaD);
            //z acceleration
            zDD = (OM/r2)*lambda*sD - MU*ze/r3;

            accel[0]= rhoDD;
            accel[1]= thetaDD;
            accel[2]= zDD;

            accel_ptr =&accel;
            return accel_ptr;
            }// end function


/*************************************************/
      double     *defectCalc(double     xLt[6],double     xRt[6],double
xLtDot[6],double xRtDot[6],double uLt[2],double uRt[2],double tSeg){
      //function  to  calculate  the  defect  values  from  the  state  and
control //vectors

      //initialize variables
            double xC[6];
            double xCDot[6];
            double uC[2];
            double stCDot[6];
            double ddC[3];
            double *ddC_Ptr;
            static double defect[6];
            double *defect_Ptr;

            int k =0;
            int i=0;
```

```
            defect_Ptr = &defect;

            for (i =0; i < 6; i++){
            // calculate xc, xcDot and uc
                  xC[i]              =    .5*(xLt[i]    +    xRt[i])    +
(tSeg/8.0)*(xLtDot[i] - xRtDot[i]);
                  xCDot[i]  =  -(3.0/(2.0*tSeg))*(xLt[i]  -  xRt[i])  -
0.25*(xLtDot[i] + xRtDot[i]);

                  if(i <2){
                  uC[i] = .5*(uLt[i] + uRt[i]);
                  }// end if
            }//end for each component

            // actually calculate current defect
            for (k = 0; k < 3; k++){
                  stCDot[k] = xC[k+3];
            }//edn for

            //state vector at center of node
            ddC_Ptr = deriv(xC,uC);

            for (i=0;i<3;i++){
                  ddC[i] = *ddC_Ptr++;
            }// end for

            for (k = 0; k < 3; k++){
                  stCDot[3+k] = ddC[k];
            }//edn for

            for (i =0; i < 6; i++){
                  defect[i] = stCDot[i]-xCDot[i];
            }// end for

            //defect[6] = uC[0];
            //defect[7] = uC[1];
      return defect_Ptr;

      }//end calc defect function
```