THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF AEROSPACE ENGINEERING

PARTICLE SWARM OPTIMIZATION APPLIED TO OPTIMIZE ORBITAL DECAY AND RE-BOOST TRAJECTORIES

JACK CHISHOLM
Spring 2012

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

Reviewed and approved* by the following:

Robert G. Melton
Professor of Aerospace Engineering
Director of Undergraduate Studies
Thesis Supervisor and Honors Advisor

David B. Spencer
Associate Professor of Aerospace Engineering
Faculty Reader

George A. Lesieutre
Professor and Head of the Department of Aerospace Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Particle Swarm Optimization is a population-based stochastic method developed in recent years and successfully applied in several fields of research. Inspired by the behavior of bird flocks while searching for food, this method is meant to use information sharing to determine a global optimum solution. For this research, particle swarm optimization is used to optimize orbital decay and re-boost trajectories for spacecraft in Low-Earth orbits. The equations of motion for such a problem include solving ordinary differential equations, which model effects of gravity, low-density drag, and thrust accelerations. The thrust trajectory is optimized so that a spacecraft in circular orbit can re-boost back to its original orbit after decaying as a result of drag. The problem is solved using MATLAB and the code includes a series of nine functions which include the ode45 built-in numerical integrator. Three possible solutions are examined for the best solution: a single continuous thrust, a two-thrust maneuver that is split by a coasting arc, and a variable magnitude trust. Each method is further examined for possible sources of error. Ultimately, the two-thrust method is determined to be the most effective because it is the least error prone and used the least amount of propellant. Using the two thrust method, other spacecraft parameters are varied which include, the coefficient of drag, the spacecraft's cross-sectional diameter, the spacecraft's mass, the effective exhaust velocity of the thruster, and the thrust-to-mass ratio of the spacecraft. The starting and minimum altitude are also varied from as high as 300 km to as low as 150 km. Future work will include tracing sources of error in particle swarm optimization and its tendency to occasionally not find the global minimum as well as to correct some data inconsistencies in the variable thrust method.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Nomenclature

A (DU^2) - the cross-sectional area of the spacecraft

B = beta - the coefficients of the thrust magnitude polynomial

BLp - the lower bounds of the particle's allowable position

BLv - the lower bounds of the particle's allowable velocity

BUp - the upper bounds of the particle's allowable position

BUv - the upper bounds of the particle's allowable velocity

c (DU/TU) - effective thrust velocity of the spacecraft's thrust

c_c = c_c - cognitive weighting coefficient

C_D - the coefficient of drag of the spacecraft

c_I = c_I - inertial weighting coefficient

conv - the conversion of density/Mass to drop the MU

c_s = c_s - social weighting coefficient

D (DU) - the cross-sectional diameter of the spacecraft

d - the magnitude by which a final condition fails to meet the requirements

DU - a distance unit in canonical units (for this report it is 1 Earth's radius)

GG - the best value of J for a given iteration

GG* or GGStar - array with the GG of each iteration

J - the summation of the errors and the time of fuel burns

JBest - the best value of J for a given particle in any iteration

K - the summation of the error terms

mass_ratio = m_f/m_0 - the final mass to initial mass ratio

$M_{s/c}$ - Mass (MU) - the initial mass of the spacecraft

MU - one mass unit in canonical units

N_elements - the number of particle variables

N_iterations - the number of iterations for the optimization

N_particles - the number of particles in PSO

n_0 (DU/TU^2) - the initial thrust-to-mass ratio of the spacecraft

ode45 - a built in numerical integrator in MATLAB

P - the two dimensional array that holds the position for each particle and all of its variables

Particle Variables – variables that are optimized by PSO (thrust polynomial coefficients, t_2, etc.)

PBest - the P position that gave the particle its best value of J

PSO – Particle Swarm Optimization

r (x_1) (DU) - the radius of the spacecraft's orbit

$\dot{r}$ = r_dot (x_2) (DU/TU) - the time derivative of the radius of the spacecraft's orbit

R_1 (DU) - the minimum allowable radius of the spacecraft's orbit

R_2 (DU) - the initial and final radius of the spacecraft's orbit

Spacecraft Variables – variables that are characteristic of the spacecraft (coefficient of drag, cross-sectional diameter, mass, effective thrust velocity, thrust-to-mass ratio)

t_1 (TU) - time of the decay

t_2 (TU) - time of the thrust

tspan - the time span for the ODE functions

TU - a time unit in canonical units

V (DU/TU) - the velocity of each particle in each of the particle variables' dimensions

v_θ - transverse velocity of the spacecraft

x - the steady state variables over a time span

x_decay - the solution to the decay steady state variables

x_save - the final position and velocities of the spacecraft after the minimum altitude is reached

zeta - the coefficients of the thrust angle polynomial

α (alpha) - the multiplicative factor for an error

θ = theta (x_3) - the angular position of the spacecraft

θ = theta_dot (x_4) - the time derivative of the angular position of the spacecraft

μ - muB (DU^3/TU^2) - the standard gravitational parameter of Earth

ρ = density (MU/DU^3) - the density of the atmosphere at a given altitude

Chapter 1

Introduction

There are many practical reasons to have satellites in low altitude orbits (150 km to 300 km). Satellites at lower altitudes have lower power costs from communication antennas and need less of a data rate. In addition, low orbit satellites provide more accurate imaging of Earth's surface. However, satellites at low altitudes are subject to high drag forces that alter spacecraft orbits and often cause reentry. As a result, re-boost maneuvers to the original orbit need to be optimized in order to increase the longevity of the mission. In this thesis, equations of motion are developed to take into consideration accelerations due to gravity, drag, and thrust to optimize the fuel consumption of a low orbit satellite. This optimization process will be done using Particle Swarm Optimization.

Initially, the satellite will be placed in a circular orbit at a user defined radius. After being subject to drag forces over a period of time, the satellite will decay to a user defined lower orbit at which point the thrusters are turned on. The objective of the spacecraft is to climb back to the original altitude using the least amount of propellant and complete the thrust maneuver in a circular orbit.

Three thrust methods are tested to achieve the best result. The first is a single and continuous magnitude thrust. The second is a two thrust maneuver with one thrust to start the maneuver, followed by a coast arc, and finally a thrust to push the spacecraft into the correct altitude and circular orbit. Finally, the third method is to have a continuous thrust for the length of the thrust maneuver but allow the thrust magnitude to vary as a function of time.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic (also referred to as evolutionary) algorithm that is designed to find a global optimum solution. Originally created in 1995 by Eberhart and Kennedy,[1] PSO is meant to mimic the unpredictable behavior of bird flocks while searching for food. Birds are referred to as particles and their position represents a possible solution.

Particles are initially randomly generated before the first iteration and are assigned a position and velocity (or position update for the next iteration). After each iteration, the best particle (or position) is selected. The velocity for the other particles are updated using a set of stochastic weights, one of which is the social component which is related to the best position visited by a portion of the particles that form the swarm.[2]

There are a number of factors that contribute to the effectiveness of the PSO algorithm. One such factor is the stochastic weights that affect the position updates of the swarm. Another is the number of particles and number of iterations. In general, increasing the number of particles and number of iterations give more accurate and reliable results but can be computationally expensive. As a result, finding the solution without increasing the number of iterations or the number of particles is often preferred. In general, PSO is a very intuitive technique, easy to program, and is often capable of finding a global minimum in a reasonable amount of time.[2]

PSO will optimize several different variables for the decay and thrust trajectories. It will first optimize the time it takes for the satellite to decay to the minimum altitude. Next, it will optimize the time of the thrust maneuver occurs and the angles at which the thrusters fires. Finally, the thrust magnitude needs to be optimized in the variable thrust case.

Review of Literature

Pontani and Conway[1] use particle swarm optimization in four different aerospace orbital transfer applications: (i) determination of the globally optimal two- and three-impulse transfer trajectories between two coplanar circular orbits; (ii) determination of the optimal transfer between two coplanar, elliptic orbits with arbitrary orientation; (iii) determination of the optimal two-impulse transfer between two circular, non-coplanar orbits; (iv) determination of the globally optimal two-impulse transfer between two non-coplanar elliptic orbits. The paper concludes that many PSO solutions are problem dependent and more difficult problems (ones that include more constraints and have a wider range of potential solutions) often need more particles to find the best solution. In addition, the paper concludes that once the particle population is significantly increased, the PSO algorithm is successful in finding the global minimum in just one code run.[1]

Another paper by Pontani and Conway[2] uses PSO in space trajectory applications and solves the following problems: the determination of periodic orbits in the context of the circular restricted three-body problem, and the optimization of (impulsive and finite-thrust) orbital transfers. From these studies, Pontani and Conway once again find that PSO gives reliable and accurate answers to the problems studied. The finite-thrust case however proves to be slightly more troublesome because the problem is more complex. They conclude that better solutions can be found using more particles, more iterations, or adjusting the weighting coefficients.[2]

Kai-Jian Zhu, Jun-Feng Li, and He-Xi Baoyin[3] study optimizing satellite maneuvers to change inclinations in sun-synchronous orbit using a combination of PSO

and Differential Evolution (DE) algorithms. The goal of the orbit change is to also identify the optimal way to save fuel and increase the amount of time a satellite is in view of a particular place on Earth (i.e. a disaster region). The paper concludes that the PSO algorithm cannot precisely confirm that the global minimum is reached (because of the problems complexity) but with the combination of the DE algorithm finds a more reliable solution.[3]

Kazuhisa Fujita and Atsushi Noda[4] study the aerodynamic disturbances on satellites that occur between the altitudes of 160 km to 300 km. In their tests, ion engines are used to maintain the altitude of their satellite which is the most fuel efficient method. They concluded that drag coefficients of the spacecraft depends very little on the altitude but more on the surface material selected.[4]

E. Doornbos, H. Klinkrad, and P. Visser[5] gain better accuracy of density models in the Earth's upper atmosphere as well as discuss the effects that air density has on satellites in low orbits. The paper concludes that satellites in circular orbits at low altitudes (150 km to 300 km) can be largely affected by air density, a parameter that can vary from 15-30%.[5]

Chapter 2

Statement of the Problem

For this problem, the goal is take a spacecraft in circular orbit starting at R_2 and, after decaying to R_1, re-boost back to the original circular orbit. The starting conditions for the initial circular orbit are

$$\begin{aligned} r_0 &= R_2 = x_1 \\ \dot{r}_0 &= 0 = x_2 \\ \theta_0 &= 0 = x_3 \\ \dot{\theta}_0 &= \sqrt{\frac{\mu}{R_2^{\,3}}} = x_4 \end{aligned} \tag{1}$$

where x_1-x_4 are the steady-state variables, μ is the standard gravitational parameter of Earth, r is the radius of the spacecraft, and ϑ is the spacecraft's true anomaly.[7] Next, the steady-state variables need to be solved. The summary of the equations of motion for the decay sequence are

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= a_{rg} + a_{rD} = \frac{-\mu + x_1(x_1x_4)^2}{x_1^{\,2}} - \frac{1}{2}\frac{C_D\rho A x_2\sqrt{x_2^{\,2} + (x_1x_4)^2}}{M} \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= \frac{a_{\theta g}}{r} + \frac{a_{\theta D}}{r} = \frac{-2x_2x_4}{x_1} - \frac{1}{2}\frac{C_D\rho A x_1x_4\sqrt{x_2^{\,2} + (x_1x_4)^2}}{x_1} \end{aligned} \tag{2}$$

where a_{rg} is the radial gravitational acceleration term, a_{rD} is the radial drag acceleration term, $a_{\vartheta g}$ is the transverse gravitational acceleration term, $a_{\vartheta D}$ is the transverse drag acceleration term, C_D is the coefficient of drag, ρ is the density of the atmosphere at a given radius, A is the spacecraft's cross-sectional area, and M is the mass of the spacecraft.[7] Note that all of these terms must be in canonical units.

The area is assumed to be circular with diameter D.

$$A = \pi \frac{D^2}{4} \tag{3}$$

After the spacecraft has decayed to R_1 using the equations of motion in Eq.(2), the re-boost sequence begins with the initial conditions set equal to the final positions and velocities of the decay sequence. The re-boost sequence will continue until the final circular orbit is achieved. Note that these equations are identical to the decay equations with the exception that there is a thrust acceleration. The equations of motion for the re-boost trajectory are

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = a_{rg} + a_{rD} + a_{rT} = a_{rg} + a_{rD} + \frac{cn_0}{(c - n_0)t} \sin\left(\zeta_0 + \zeta_1 t + \zeta_2 t^2 + \zeta_3 t^3 + \zeta_4 t^4 + \zeta_5 t^5\right)$$

$$\dot{x}_3 = x_4 \tag{4}$$

$$\dot{x}_4 = \frac{a_{\theta g}}{r} + \frac{a_{\theta D}}{r} + \frac{a_{\theta T}}{r} = \frac{a_{\theta g}}{r} + \frac{a_{\theta D}}{r} + \frac{cn_0}{(c - n_0)x_1 t} \cos\left(\zeta_0 + \zeta_1 t + \zeta_2 t^2 + \zeta_3 t^3 + \zeta_4 t^4 + \zeta_5 t^5\right)$$

where a_{rT} is the radial thrust acceleration term, $a_{\vartheta T}$ is the transverse thrust acceleration term, c is the effective thrust velocity, n_0 is the initial thrust-to-mass ratio, ζ_0- ζ_5 are the thrust angle polynomial coefficients, and t is the time of the thrust maneuver.

The errors need to be calculated next to see how close the spacecraft is to meeting its required final conditions. These conditions are

$$\dot{r}_f = x_{2f} = 0$$

$$v_{\theta f} - v_{\theta i} = x_{1f} x_{4f} - \sqrt{\frac{\mu}{R_2}} = 0 \tag{5}$$

$$r_f - r_i = x_{1f} - R_2 = 0$$

Once the errors are calculated, the final mass to initial mass ratio is

$$\frac{m_f}{m_0} = 1 - \left(\frac{n_0}{c}\right)\sum t_{Thrust} \tag{6}$$

where m_f is the final mass, m_0 is the initial mass, and t_{Thrust} is any period of time when the spacecraft is in a thrust sequence (this accounts for multiple thrust sequences). Finally, the total amount of fuel consumed can be calculated using

$$m_p = m_0\left(1 - \frac{m_f}{m_0}\right) \tag{7}$$

where m_p is the mass of the propellant (fuel used).

Additionally, the above equations need to be adjusted in the event that the thruster has the capability to vary the propellant use in a thrust sequence. The radial and thrust acceleration components are modeled as

$$\begin{aligned} a_{rT} &= \frac{cn_0\left(\beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \beta_5 t^5\right)}{\left(c - n_0\left(\beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \beta_5 t^5\right)\right)t}\sin\left(\zeta_0 + \zeta_1 t + \zeta_2 t^2 + \zeta_3 t^3 + \zeta_4 t^4 + \zeta_5 t^5\right) \\ a_{\theta T} &= \frac{cn_0\left(\beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \beta_5 t^5\right)}{\left(c - n_0\left(\beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \beta_5 t^5\right)\right)x_1 t}\cos\left(\zeta_0 + \zeta_1 t + \zeta_2 t^2 + \zeta_3 t^3 + \zeta_4 t^4 + \zeta_5 t^5\right) \end{aligned} \tag{8}$$

where B_0-B_5 are the thrust magnitude polynomial coefficients. Also, an additional error needs to be considered because the thrust magnitude can never exceed 1 or be less than 0.

$$0 \le \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \beta_5 t^5 \le 1 \text{ for all t} \tag{9}$$

As a result of the variable thrust booster, the mass ratio equation is changed to reflect the proper fuel consumption

$$\frac{m_f}{m_0} = 1 - \left(\frac{n_{0avg}}{c} \right) t_{Thrust} \tag{10}$$

where n_{0avg} is the thrust-to-mass ratio multiplied by the average value of the thrust magnitude polynomial over a given time period t.

Atmospheric Model

In order to determine the atmospheric density at a given altitude, data from MSISE-90 Model of Earth's Upper Atmosphere is plotted with lines of best fit. The data are originally given in 20 km increments so a best fit line gives a good estimate of the density between those intervals. The data specified are measured at an average solar radiation level. For this set of experiments, the altitudes will range from 150 km – 500 km. An example of a plot from 200 km to 300 km is provided below in Figure 2-1 with its line of best fit and R^2 value. The remaining plots and the table of data are provided in Appendix 1 at the end of the report.

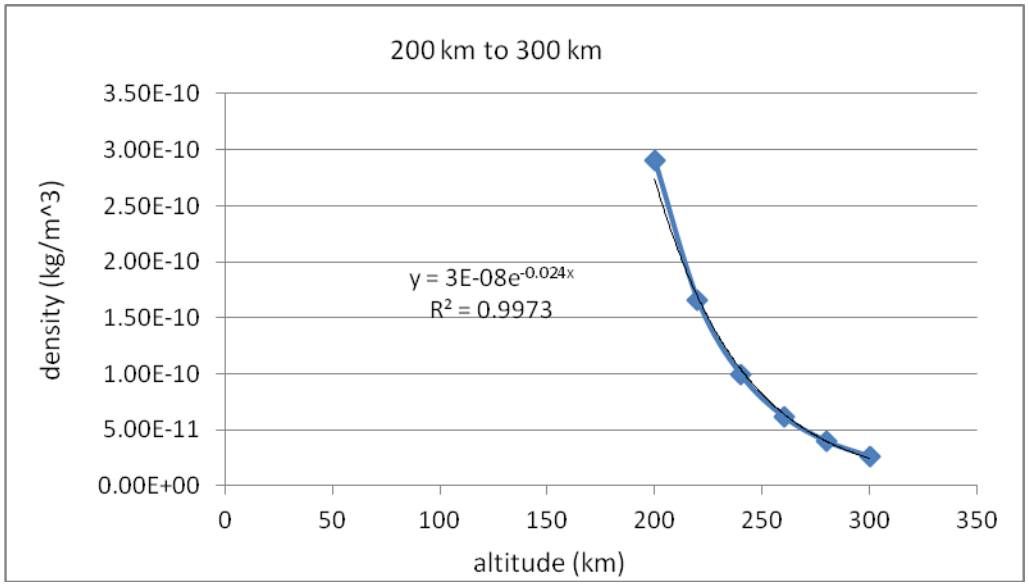


Figure 2-1 Altitude vs. Density Example Plot (200 km – 300 km)

Particle Swarm Equations

The first portion of the particle swarm optimization algorithm is to determine the factors that comprise performance of the tested solution or J. J is a function of the errors in the required final conditions and the total amount of fuel burned. In this case, the errors, which are shown in Eq.(5) (and Eq.(9) if the thrust magnitude can be varied), are defined by the variable d. Next, a weighting factor needs to be determined to place more prominence on the error and this weighting factor is given the variable α. As a result, the effects of d and α combine to create the error

$$K = \sum |d| \alpha \tag{11}$$

Now that the error term K is calculated, J can be calculated using the error term and the time of thrusts. This relationship is

$$J = K + \sum t_{fuel} \tag{12}$$

where t_{fuel} is the length of time fuel is used. Note that there is a key difference for the variable thrust magnitude option. The J function will have a slightly different calculation because the time of thrust will be multiplied by the average thrust magnitude

$$J = K + \sum t_{fuel} n_{avg} \tag{13}$$

Once J is calculated for all particles, the best particle can be chosen and labeled G_{Best}. In addition, the particle should also compare its J value to its best J value and label its position P_{Best}. The next step would be to update the velocity of the particle using P_{Best}, G_{Best}, the particle's previous position P, and the particle's previous velocity V. The equation for the new velocity V_{new} is

$$V_{new} = c_I V + c_c \left(P_{Best} - P\right) + c_S \left(G_{Best} - P\right) \tag{13}$$

where c_I is the inertial weighting coefficient, c_c is the cognitive weighting coefficient, and c_s is the social weighting coefficient. These three terms are defined as

$$c_I = \frac{(1+rand)}{2}$$
$$c_c = 1.49445 rand \tag{14}$$
$$c_s = 1.49445 rand$$

where *rand* is a random number uniformly distributed between 0 and 1. Now with an updated velocity, the particle's position for the next iteration can be calculated using

$$P_{new} = V_{new} + P \tag{15}$$

Chapter 3

Method

The Method section describes the algorithm and the nine m-files that complete the code used to solve this problem. Note that the code being described is for the single continuous magnitude trust code. In addition, this section will describe any differences between the single thrust and the two thrust or variable-magnitude thrust codes. All three sets of code can be found in the Appendix. Finally, this section will be summarized with commonly found run-time errors and how they can be avoided.

PSOTest_adaptive (Main)

PSOTest_adaptive.m is the main function for the code. The function itself is broken down into 6 subcomponents: variable definition, initialize random population, particle bounds, velocity bounds, solve, and printing and plotting final results.

The variable definition portion defines and allows for the user to adjust any spacecraft, orbit, or PSO variables. PSO variables are defined first which include the N_particles (number of particles), N_elements (number of variable elements the PSO is optimizing) and N_iterations (the number of iterations the code will complete). For the purposes of this thesis, N_particles=50, N_elements=7 (6 for the thrust polynomial and 1 for t2), and N_iterations=200 (the code generally found the optimal solution in about 100 iterations but is sometimes as high as 170 iterations). The spacecraft and orbit conditions are also defined here such as t1, μ, c, n_0, $M_{s/c}$, C_D, D, A, R_1, and R_2. The code is commented to indicate the units required for each variable input.

The initialization of the random particle population is the next component of PSOTest_adaptive. Using the "rand" function in Matlab, the population is first initialized with the particle elements uniformly distributed within the corresponding boundaries. All thrust coefficient variables must be within -1 and 1. Thrust bounds are set between 10^{-3} and 3 TU. Finally, memory is allocated for the PBest, J, JBest, and V arrays and set to zero.

The particle bounds are defined next according to the same boundaries described in the particle initialization. BLp and BUp are the arrays that set the lower bounds and upper bounds for the particle variables respectively.

The velocity bounds portion determines the maximum (BUv) and minimum (BLv) velocity a particle can have. Also, the JBest array and GG* variable is set to infinity to ensure that a better solution will be found.

Next the Solve portion calls a variety of functions to determine the optimal solution for each iteration. First, the Decay function is called which determines t1 and the spacecraft movement during the decay from R_2 to R_1. Next, the main loop of the code starts from 1 to N_iterations. The loop prints out the iteration number, determines J for all particles, determines PBest and GG*, updates the velocity of the particles, updates the position of the particles, and finally prints GG*, t_1, t_2, and the mass ratio.

Lastly, the PSOTest_adaptive code calls the FinalResults function which determines the spacecraft mechanics for the optimum solution and prints out the data and plots that describe its trajectory in metric units.

Decay

The Decay function determines the spacecraft motion during the decay sequence and therefore is significantly influenced by atmospheric drag. First, the decay function initializes the decay differential equations by setting the initial conditions to a spacecraft in circular orbit starting at the higher radius R_2. The time span encompasses from 0 to t_1 TU. The differential equations of motion are solved using MATLAB's ode45 function with absolute and relative tolerances of 10^{-8}. Once the ode45 completes, the decay function determines at which time the spacecraft's radius reaches the minimum allowable radius or R_1. r, r_dot, theta, and theta_dot are all saved as a function of time in canonical units as an array called x_save. Finally, t_1 is updated to reflect the time when the lower radius is met.

EvalJ

EvalJ is the function that calculates J for all of the particles. Remember that J is a function of the thrust used and its ability to meet the final conditions of a circular orbit with radius R_2. EvalJ begins with a loop that will run the following calculations on each particle. The seven variables that are to be optimized are read into the function and renamed to zeta0-zeta5 and t2. EvalJ then takes the final conditions of the decay differential equations and sets them to the initial conditions for the thrust differential equations. Once again tolerances are set but this time to 10^{-6} because this is the section to be iterated on and will therefore take much more time. The thrust ode45 function is then called and returns the spacecraft's motion as a function of time.

Next, the EvalJ function determines the errors of the spacecraft's final position and velocity by determining how close the spacecraft meets the necessary final conditions of a

circular orbit at radius R_2. To meet the conditions of a circular orbit, the three potential errors are checked to a tolerance of 10^{-3}.

In the event that the final conditions are not within the tolerances specified, the error is multiplied by 100 (otherwise known as the variable alpha) and stored in the array K. J is then calculated as the sum of t_2 and K. The EvalJ process is repeated for all particles in each iteration.

EvalPGBest

EvalPGBest determines which particle performed the best or in other words had the lowest J value. PBest is also calculated which is simply the values of P (or the optimization variables) for the particle that had the lowest J value. Next GG* is calculated which is the lowest value of J for all iterations. The spacecraft mass-ratio (m_f/m_0) is calculated and stored which is later output after the end of each iteration. Finally, the best particle is stored by its number.

UpdateV

UpdateV updates the velocity of the particles using variable accelerator coefficients. The variable coefficients are set to the same values as Pontani and Conway used in their paper Particle Swarm Optimization Applied to Space Trajectories[2] which are defined in Chapter 2. Once the accelerator coefficients are determined for a given iteration, the velocity of each particle is calculated. Each variable that is to be optimized is then tested to see if the velocity is within the bounds set in PSOTest_adaptive. In the event that the particle velocity is outside of the bounds set, then the velocity is changed to the maximum allowable velocity (if the velocity is too large) or minimum allowable velocity (if the velocity is too small).

UpdateP

UpdateP updates the position of the particles using the particle's current position and its velocity which is calculated in UpdateV. The new position for each particle is calculated first and is simply the sum of the particle's former position and its velocity. In the event that the particle's new position is outside the allowable boundary set in PSOTest_adaptive, the particle will position itself at the closest position on the boundary.

FinalResults

FinalResults is a function that takes the best particle during the final iteration and prints various characteristics of the spacecraft's motion. First, FinalResults takes the best particle, re-runs the thrust maneuver, and stores all of its position and velocity data. Arrays are created to plot the spacecrafts altitude vs. time and altitude vs. theta for both the decay and thrust sequences in metric units. Ultimately, five plots are created. The first shows the convergence of GG* vs. iteration number to show that the best solution is generally found within the number of iterations specified by the user. This can be confirmed when there is no more decrease in GG* after a certain point. Next, the altitude vs. theta followed by the altitude vs. time are plotted for both the decay and thrust time periods. Finally, a number of other features for the trajectory are calculated and printed which includes the spacecraft's final altitude, the number of revolutions of decay and thrust, the fuel used in kg , the fuel used per revolution, and the time of the cycle (days), decay (days), and thrust (minutes).

PSOode

The PSOode, otherwise known as the decay ODE, calculates the spacecraft's trajectory during the orbital decay portion of the cycle. The spacecraft starts with the initial conditions set in the Decay function and continues its decay until the time reaches t_1. Using the atmospheric model developed in Chapter 2, the density at the spacecrafts radius is calculated for each given time. Next the equations of motion are calculated for the decay which include only the gravitational and drag accelerations. Once completed the PSOode returns the spacecraft's motion for the decay period as a function of time. This ODE is set to a tolerance of 10^{-8} and is run only once but can take up to several minutes to complete.

PSOode2

The PSOode2, otherwise known as the thrust ODE, calculates the spacecraft's trajectory during the thrust portion of the cycle. The algorithm for this ODE is similar to PSOode but with a few key differences. The spacecraft starts with the initial conditions set by the end of the decay trajectory as stated in the EvalJ function. In this ODE the spacecraft is constantly changing its mass which is accounted for in the drag acceleration term. Also, the thrust acceleration term is added to the equations of motion which include the thrust direction polynomial. This ODE is calculated to a tolerance of 10^{-6} and is run for each particle and for each iteration but takes a split second to complete.

Changes for Two Thrusts

In order to accommodate two thrusts instead of a single continuous thrust, several changes need to occur. The number of elements increases to 15, with 6 for the first thrust angle polynomial, 6 for the second thrust angle polynomial, and 3 for t_2 (first thrust time), t_3 (coast time between thrusts), and t_4 (second thrust time). The particle bounds arrays for position and velocity need to be changed accordingly. In the EvalJ function, two more ODEs need to be added in addition to a variable that will keep track of the total mass of the spacecraft as fuel is burned. There will be two PSOode2's for the thrusts separated by a PSOode for the coast that occurs in between. Also, J will include t_4 as part of the sum. In EvalPGBest, the mass ratio equation will also need to be adjusted to represent the propellant used in the two burns. FinalResults will see similar changes as EvalJ does, as well as changes in the output so that both thrusts and the coast time are displayed.

Changes for Variable Thrust Magnitude

In order to accommodate a variable magnitude thrust instead of a single continuous thrust, several changes need to occur. The number of elements increases to 13, 6 for the first thrust angle polynomial, 6 for the thrust magnitude polynomial (called beta), and 1 for t_2. The particle bounds arrays for position and velocity need to be changed accordingly and the range for beta is [-1,1]. In the EvalJ function, beta0-beta5 need to be included and passed to PSOode2. J will have two more potential errors: an error for if the thrust polynomial exceeds 1 and an error if the thrust polynomial is less than 0. Also the average thrust magnitude needs to be calculated and multiplied by t_2 to reflect the amount of propellant used in J equation. PSOode2 also will

undergo serious changes because the thrust magnitude (n_0) will be multiplied by the thrust magnitude polynomial. EvalPGBest will need to have the average thrust magnitude to calculate the mass ratio. FinalResults will see similar changes as EvalJ to correctly calculate the propellant mass used. Also, the thrust magnitude polynomial should be plotted to ensure that the thrust magnitude never exceeds 1 or is less than 0.

Common Run-Time Errors

Three problems occur on occasion with the code. The first is an actual error that states that x_save is unable to be found. This occurs when the guess for t_1 is too small which results in the spacecraft never decaying to the minimum specified radius. Increasing t_1 and rerunning the code should alleviate this problem.

The second problem occurs when the Decay function takes too long to complete the PSOode. This occurs when the guess for t_1 is too large which causes the spacecraft to descend too far. This significantly slows the calculation for the equations of motion to meet the tolerance. If there is ever the concern that this occurred, stop the code from running, reduce the value of t_1 and rerun. This can be an especially large problem because runs at high altitudes generally take a considerable amount of time to calculate. A recommendation for high altitude testing would be to print output in the PSOode function to ensure that the code is still running quickly.

The final problem is a general issue that sometimes occurs in PSO. On occasion, the initially randomized particles will find a local minimum and converge to that point. A safe testing protocol would be to run each test at least twice and take the result that has no errors and the minimum propellant used. Sometimes this issue can be resolved in mid-run if the user notices the code is running up on one of the time boundaries. If this occurs, stop the code and rerun. Extra runs of a single case found that the fuel used can vary as much as 20%.

Chapter 4

Results

The Results section describes the tests run on the code, displays their results, and discusses the various findings from the results. A total of seven tests are completed. The first test determines which method (i.e. single continuous thrust, two thrusts, or variable magnitude thrust) used the least amount of propellant and had the least number of errors. Once the best method is chosen, tests are conducted on factors such as the altitude (test 2), coefficient of drag (test 3), cross-sectional diameter (test 4), spacecraft mass (test 5), effective thrust velocity (test 6), and initial thrust-to-mass ratio (test 7) by varying each individual variable while keeping all others the same.

To ensure that the data is as accurate as possible, each test is run without restarting MATLAB. Each condition is run only once primarily to finish the testing in a timely manner. Each run took about 15 minutes which is for 50 particles and 200 iterations. As explained in possible run-time errors, particle swarm optimization occasionally converges to a local minimum without finding the best solution. As a result, some runs may have solutions that are as much as 20% off.

Method Determination

The first test conducted is to see which of the three methods (1. single continuous thrust, 2. two thrusts, or 3. variable magnitude thrust) uses the least amount of fuel per cycle and has the fewest errors. Errors include: not finishing the thrust maneuver at radius R_2, having a non-zero radial velocity after the maneuver, and not having transverse velocity that is equivalent to a circular orbit. For the variable thrust code, the additional error is measured: the thrust magnitude

polynomial must never exceed 1 or be less than 0. All three tests include the following spacecraft variables as seen in Table 4-1 while the starting and minimum altitude vary.

Table 4-1 Spacecraft Variables for Tests 1 and 2

Condition	Value
Coefficient of Drag, Cd	2.5
Cross-Sectional Diameter, D	5 m
Spacecraft Mass, Mass	50000 kg
Effective Thrust Velocity, c	0.5 DU/TU
Thrust-to-Mass Ratio, n0	0.05 DU/TU^2

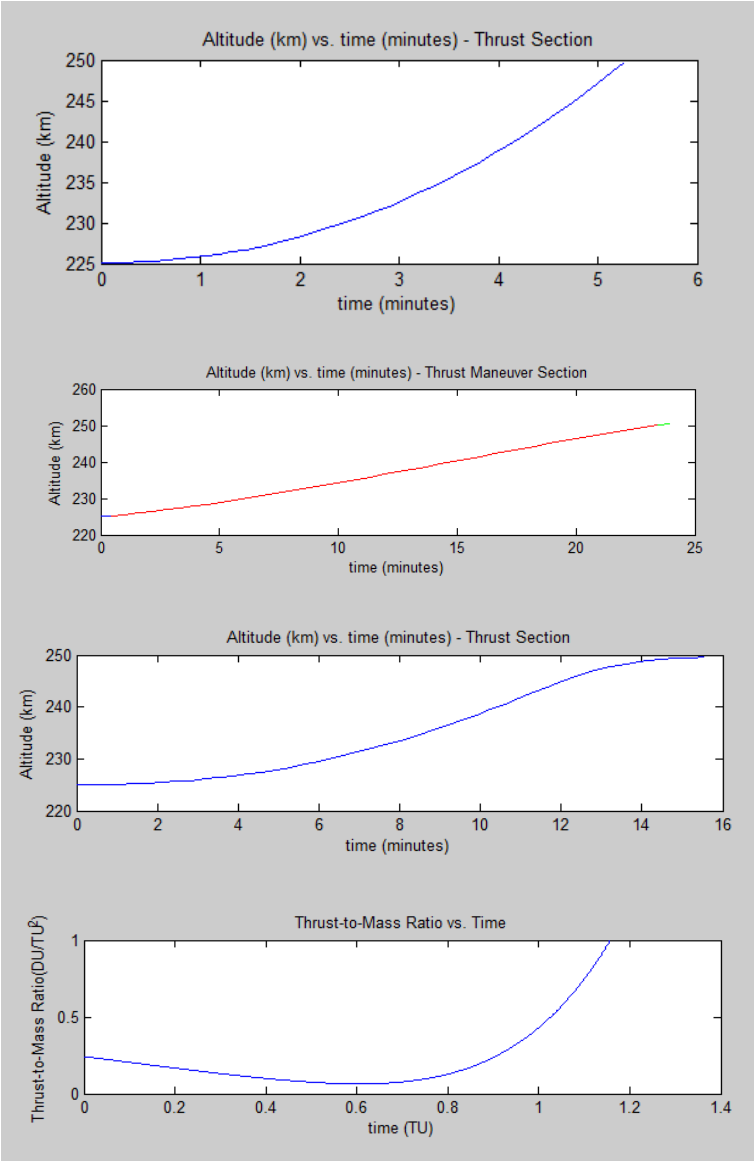
The results are show in Table 4-2 below which includes the fuel used and the number of errors. Once again the methods are as follows: 1. single continuous thrust, 2. two thrusts, or 3. variable magnitude thrust.

Table 4-2 Method Determination

Method	Starting Altitude (km)	Minimum Altitude (km)	Fuel Used (kg)	Errors
1	200	150	2964.468	2
2	200	150	337.748	0
3	200	150	1730.751	1
1	225	200	1956.261	1
2	225	200	92.132	0
3	225	200	702.083	2
1	250	225	1959.264	1
2	250	225	364.108	0
3	250	225	1283.691	0
1	250	200	8241.090	0
2	250	200	358.942	0
3	250	200	1569.531	2

.

Figure 4-1 below shows the altitude of the spacecraft as a function of time for the entire thrust maneuver for each of the methods. All plots are taken for the 225 km to 250 km case and only the single continuous thrust had an error (final radial velocity is too great). The thrust-to-mass ratio for the variable thrust is also shown in Figure 4-1.



Method 1:
Single Continuous Thrust

Method 2:
Two Continuous Thrusts
(Blue – Thrust 1,
Red – Coast between Thrusts
Green – Thrust 2)

Method 3:
Variable Magnitude Thrust

Method 3:
Variable Magnitude Thrust
Thrust to Mass ratio plot

Figure 4-1 Method Determination Plots

From a fuel consumption standpoint, method 1: single continuous thrust used the most fuel and is error prone. The most common error for method 1 is the final transverse velocity being too large. Because of the continuous thrust, the spacecraft often enters the final altitude at a velocity too large. Reducing the thrust-to-mass ratio made this error less dramatic.

Overall, method 2: Two continuous thrusts proved to be the best method because for each range of altitudes it used the least amount of propellant and never had an error.

Although method 3 (variable magnitude thrust) uses less fuel than the single-continuous thrust, it often is more error prone than the other two methods. Most of the errors are attributed to the final transverse velocity being too large or the thrust magnitude polynomial exceeding 1. This method uses twice the amount of propellant as method 2 does as well.

As a result of this test, method 2 (two thrusts) is chosen as the appropriate method for the remainder of the tests because it proved to be the most robust and most fuel efficient method for satellite re-boost maneuvers.

Altitude Variation

The next test is used to examine the effects of various altitudes on the same spacecraft. Once again, the spacecraft will maintain the characteristics mentioned in Table 4-1. Many altitude variations are considered and had decays from as little as 25 km to 100 km. The results for the test can be seen below in Table 4-3.

Table 4-3 Altitude Variation

Starting Altitude (km)	Minimum Altitude (km)	N revolutions Decay	N revolutions Thrust Maneuver	Time per Cycle (days)	Fuel Used (kg)	Fuel Used per Rev (kg)
175	150	81.946	0.399	5.020	149.982	1.821
200	175	221.035	0.532	13.584	125.362	0.566
200	150	303.243	0.562	18.597	337.748	1.112
225	200	516.846	0.497	31.892	92.132	0.178
225	175	738.713	0.508	45.495	326.073	0.441
250	225	934.590	0.269	57.959	364.108	0.389
250	200	1452.565	0.472	89.903	358.942	0.247
275	250	1689.311	0.483	105.358	96.290	0.057
275	200	3145.540	0.479	195.459	472.323	0.150
300	275	3045.677	0.485	191.564	95.564	0.031
300	250	4753.048	0.509	297.458	290.407	0.061
300	200	6208.043	0.519	387.483	657.779	0.106

Table 4-3 shows a number of interesting results from this series of tests. First, the number of revolutions of decay and time of the cycle are heavily dependent on the starting and minimum altitudes chosen. This is mostly attributed to the lower density at higher altitudes. The fuel used also changes dramatically with altitudes which show that the drag from the density does play into how the spacecraft re-boosts to the original altitude. The fuel used per revolution shows the efficiency of the cycle and is significantly lower (more efficient) for the higher altitudes than for the lower altitudes.

Perhaps the most interesting of all of these numbers is that short re-boosts are more efficient than long ones. For example, if you sum the fuel used for the 200 to 250 km, 250 to 275 km, and 275 to 300 km, the sum would be 550.796 kg of fuel used. Compare that to the 657.779 kg of fuel it takes the spacecraft to go from 200 to 300 km. This result shows that smaller climbs are often more efficient than long ones. This would require a more complicated re-boost trajectory with many small thrusts but could overall save more fuel.

Coefficient of Drag Variation

The coefficient of drag test describes how the coefficient of drag affects the decay and re-boost of the spacecraft. The spacecraft variables used for this and the remainder (Tests 3-7) can be seen below in Table 4-4.

Table 4-4 Spacecraft Variables for Tests 3-7

Condition	Value
Coefficient of Drag, Cd	2.5
Cross-Sectional Diameter, D	5 m
Spacecraft Mass, Mass	50000 kg
Effective Thrust Velocity, c	0.5 DU/TU
Thrust-to-Mass Ratio, n0	0.05 DU/TU^2
Initial/Final Altitude	250 km
Minimum Altitude	200 km

Once again, the coefficient of drag is varied in this test and the results can be seen below in Table 4-5.

Table 4-5 Coefficient of Drag Variation

Cd	N revolutions Decay	N revolutions Thrust	Time per Cycle (days)	Fuel Used (kg)	Fuel Used/Revolution (kg)
1.0	3634.252	0.513	224.887	285.873	0.079
1.5	2422.804	0.472	149.934	295.520	0.122
2.0	1817.115	0.602	112.466	349.036	0.192
2.5	1452.565	0.472	89.903	358.942	0.247
3.0	1211.438	0.562	74.989	302.880	0.250

Notice that there is no noticeable correlation between the fuel used and the coefficient of drag. This is most likely attributed to error in particle swarm optimization and the code finding a local minimum and not the absolute minimum. Despite this, the time per cycle vs. coefficient of drag showed an inverse relation that is consistent with atmospheric drag theory. The plot is in Figure 4-2 below and has a line fitted to the data.

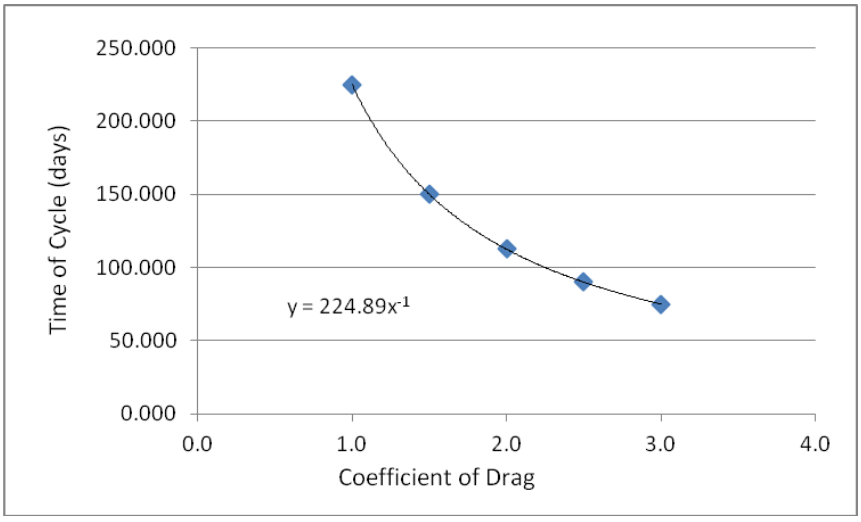


Figure 4-2 Coefficient of Drag vs. Time of Cycle Plot

Cross-Sectional Diameter Variation

The next test varies the cross-sectional diameter of the spacecraft using the spacecraft variables in Table 4-4. The results for this test are found below in Table 4-6.

Table 4-6 Cross-Sectional Area Variation

Cross Sectional Diamter (m)	N revolutions Decay	N revolutions Thrust	Time per Cycle (days)	Fuel Used (kg)	Fuel Used/Revolution (kg)
3	4037.275	0.504	249.827	286.659	0.071
4	2271.040	0.508	140.546	294.778	0.130
5	1452.565	0.472	89.903	358.942	0.247
6	1009.560	0.520	62.496	286.087	0.283
7	741.723	0.508	45.923	288.441	0.389
8	567.888	0.414	35.162	351.892	0.619
9	448.688	0.538	27.795	293.692	0.654
10	363.390	0.531	22.517	294.616	0.810

The number of days per cycle is plotted against the spacecraft-cross sectional diameter to result in an inverse squared relationship. This would conclude that cross-sectional area is an inverse relation to days per cycle. The plot with a best fit curve can be seen below in Figure 4-3.

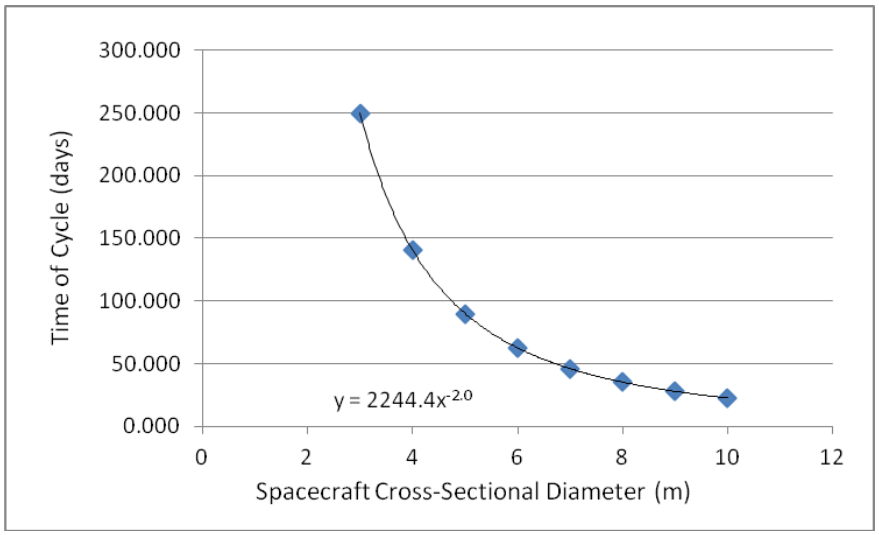


Figure 4-3 Diameter vs. Time of Cycle Variation Plot

Another characteristic to examine is the fuel used per cycle as a function of diameter. The plot, which can be seen below in Figure 4-4, shows that there seems to be little variation between the fuel used and the diameter of the spacecraft. This is most likely because the acceleration due to thrust is significantly higher than the acceleration due to drag.

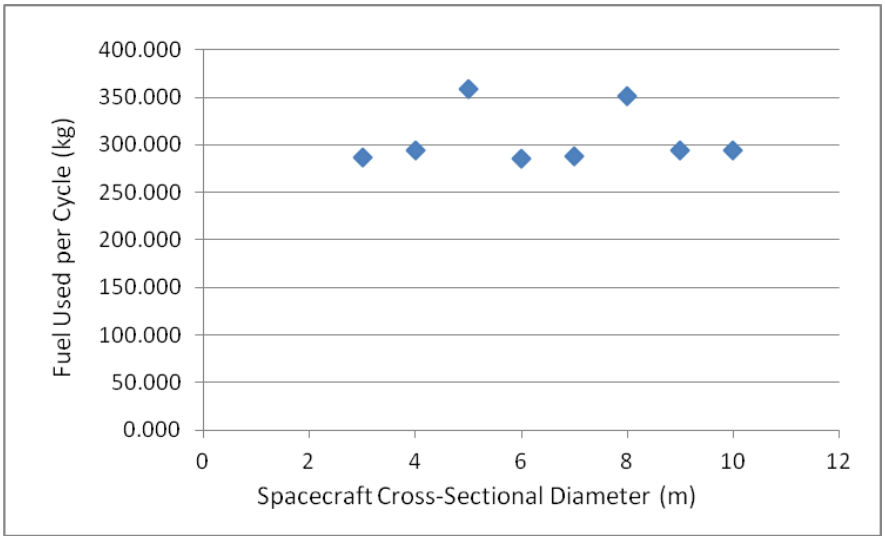


Figure 4-4 Diameter vs. Fuel Used Plot

Mass Variation

The spacecraft mass variation is the fifth test and once again uses the spacecraft variables in Table 4-4. The data results are below in Table 4-7.

Table 4-7 Spacecraft Mass Variation

Mass of Spacecraft (kg)	N revolutions Decay	N revolutions Thrust	Time per Cycle (days)	Fuel Used (kg)	Fuel Used/Revolution (kg)
20000	581.425	0.477	36.004	114.244	0.196
40000	1162.788	0.452	71.972	298.321	0.256
50000	1452.565	0.472	89.903	358.942	0.247
60000	1744.253	0.520	107.953	355.120	0.204
80000	2325.858	0.511	143.938	513.557	0.221
100000	2906.829	0.450	179.880	588.299	0.202

There are a number of interesting relationships from this table. The first is the time of cycle as a function of the spacecraft mass which is completely linear. The plot is shown below with a best fit curve in Figure 4-5. The reason the time of cycle increases with mass is because the acceleration due to drag is a function of the inverse of the spacecraft mass.

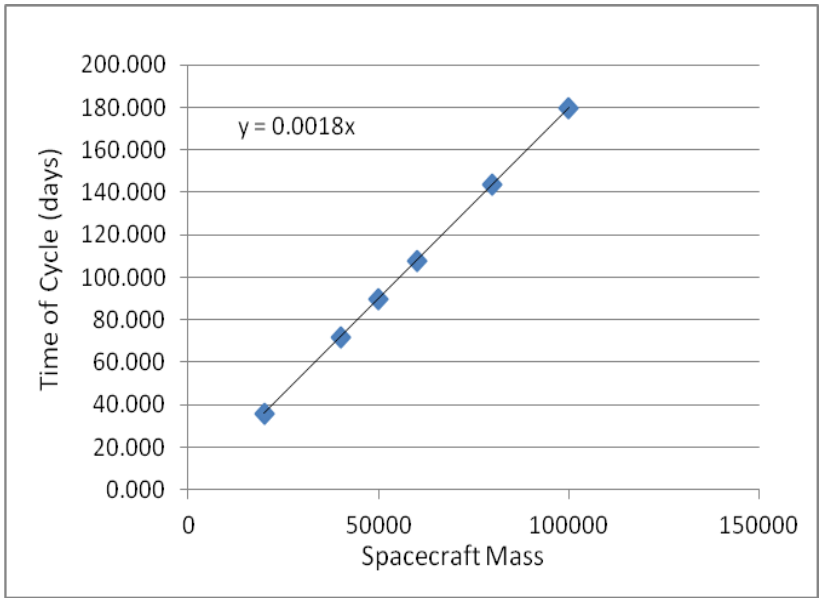


Figure 4-5 Spacecraft Mass vs. Time of Cycle Plot

The next important plot is the relationship between the fuel used per cycle and the spacecraft mass. This plot is relatively linear and increases because the more the spacecraft weighs, the more force the propellant needs to exert to overcome the forces of gravity to get back to the original orbit. This plot can be seen below in Figure 4-6.

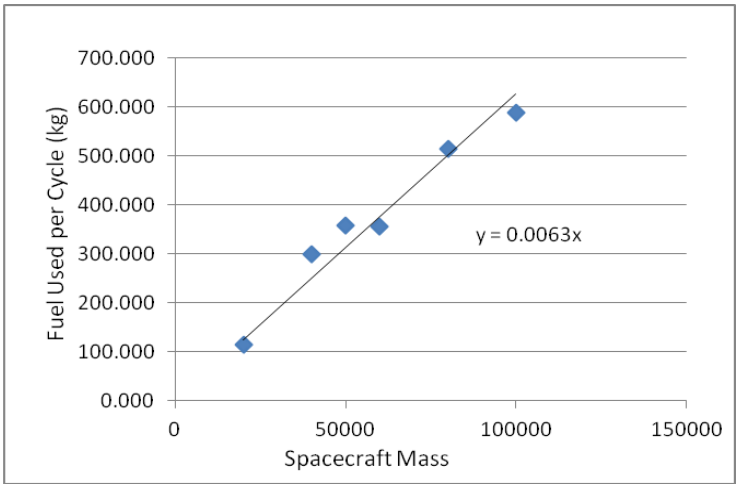


Figure 4-6 Spacecraft Mass vs. Fuel Used per Cycle Plot

As a result of Figure 4-5 and Figure 4-6, higher mass increases the time of cycle and the fuel used per cycle. The next question then is which mass gives the optimal fuel used per a given number of revolutions. Spacecraft with low masses will decay quickly but use much less propellant to regain its original orbit. Massive spacecraft are the exact opposite. As a result, over a lengthy period of time the fuel used will essentially be equal for the same spacecraft of two different masses. A spacecraft with less mass with perform more cycles over the time period than the more massive spacecraft. The results are shown in Figure 4-7 below.

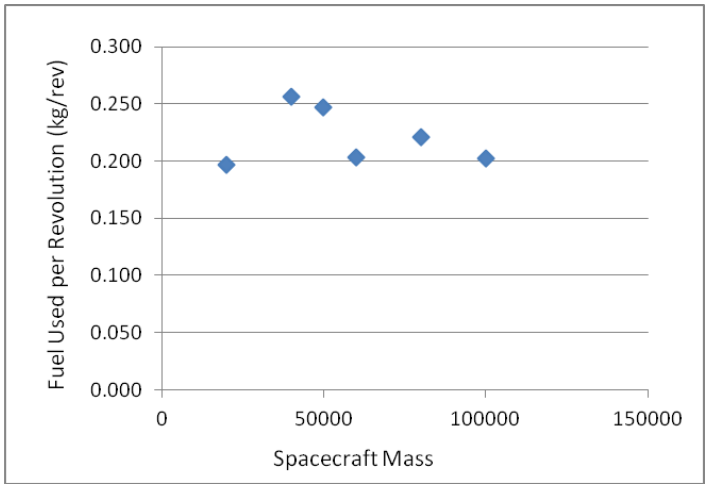


Figure 4-7 Spacecraft Mass vs. Fuel Used per Revolution Plot

Effective Thrust Velocity Variation

The next variable tested is the effective thrust velocity of the propellant. The higher the thrust velocity is, the more efficient the thrust is. Once again this test uses the spacecraft variables in Table 4-4. The data results are below in Table 4-8.

Table 4-8 Effective Thrust Velocity Variation

Exhaust Velocity (DU/TU)	Exhaust Velocity (km/s)	N revolutions Thrust	Time of Thrust Maneuver (minutes)	Time of Thrust 1 (minutes)	Time of Thrust 2 (minutes)	Fuel Used (kg)	Fuel Used per Revolution (kg)
0.2	1.581	0.472	42.037	0.498	0.272	715.1	0.492
0.3	2.372	0.495	44.047	0.497	0.287	485.6	0.334
0.4	3.162	0.432	38.480	0.598	0.282	408.8	0.281
0.5	3.953	0.472	42.037	0.597	0.368	358.9	0.247
0.6	4.743	0.472	42.032	0.501	0.266	237.6	0.163
0.7	5.534	0.501	44.596	0.497	0.280	206.5	0.142
0.8	6.324	0.502	44.699	0.500	0.267	178.2	0.123
0.9	7.115	0.424	37.761	0.562	0.323	182.7	0.126
1.0	7.905	0.500	44.535	0.497	0.281	144.6	0.099

Notice the relationship between the time of thrust maneuver and the exhaust velocity. In general, the time of the thrust maneuver is relatively similar because the time of the coasting arc is longer than the time of the thrusts themselves. Another interesting relationship is shown below in Figure 4-8 which plots the effective thrust velocity vs. the fuel used. A line of best fit is applied to this and shows that high exhaust velocities make the thrust maneuver more efficient.

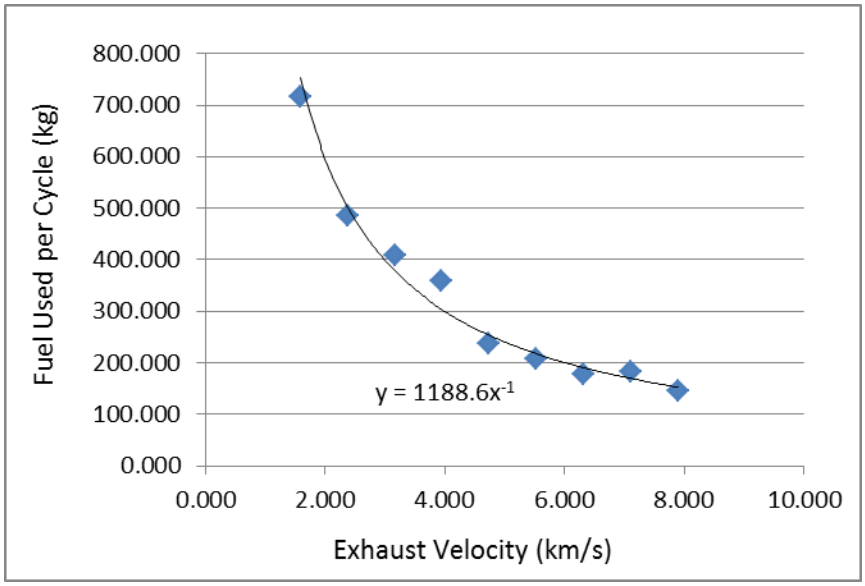


Figure 4-8 Effective Thrust Velocity vs. Fuel Used Plot

Thrust-to-Mass Ratio Variation

The final variable tested is the thrust to mass ratio. Table 4-4 is used once more as the standard spacecraft variables and the results for the test are shown below in Table 4-9.

Table 4-9 Initial Thrust-to-Mass Ratio Variation

Thrust-to-Mass Ratio (DU/TU^2)	N revolutions Thrust	Time of Thrust Maneuver (minutes)	Time of Thrust 1 (minutes)	Time of Thrust 2 (minutes)	Fuel Used (kg)	Fuel Used per Revolution (kg)
0.0010	0.615	54.661	28.388	0.001	211.1	0.145
0.0025	0.553	49.147	10.034	0.001	186.5	0.128
0.0050	0.524	46.653	4.980	0.001	185.	0.127
0.0100	0.517	46.035	2.509	1.366	288.2	0.198
0.0250	0.414	36.842	1.339	0.594	358.7	0.247
0.0500	0.472	42.037	0.597	0.368	358.9	0.247
0.0750	0.434	38.610	0.358	0.189	304.9	0.210
0.1000	0.375	33.384	0.305	0.227	395.6	0.272
0.1500	0.428	38.069	0.174	0.149	360.4	0.248

First, attention should be brought to the time of the thrust maneuver which in general decreases. This is most likely because large thrust-to-mass ratios allow the spacecraft to gain altitude much quicker because of the higher velocity change. In addition, the fuel used per cycle is generally less for lower thrust-to-mass ratios which concludes that lowering this variable will make the maneuver more efficient. The time of the second thrust shows that for low thrust-to-mass ratios, the second thrust is unneeded to meet the final conditions while higher thrust-to-mass ratios need the second thrust to make corrections for the final orbit. Finally, Figure 4-9 below shows the thrust-to-mass ratio vs. the time of the first thrust and includes a line of best fit.

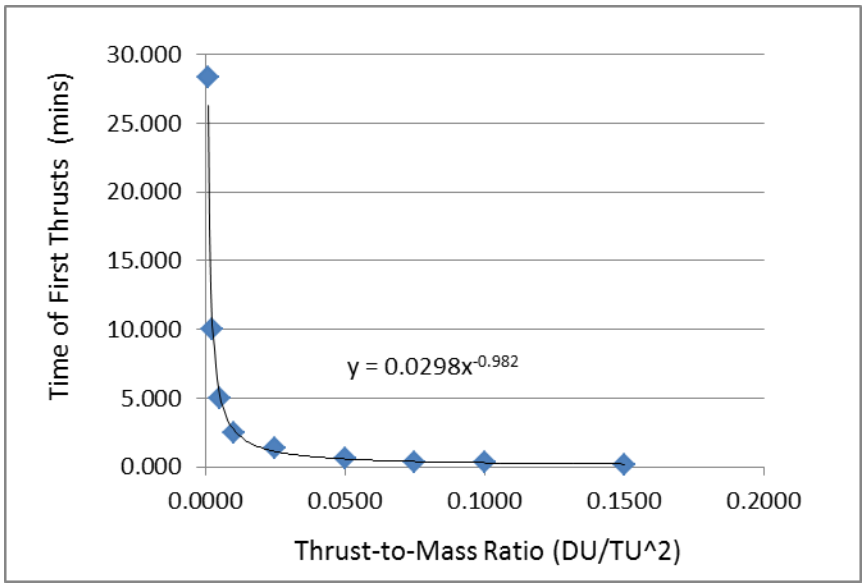


Figure 4-9 Thrust-to-Mass Ratio vs. Time of First Thrust Plot

The time of the first thrust is significantly larger at low thrust-to-mass ratios because the spacecraft takes longer to gain the velocity necessary to reach the higher altitude and circular orbit. As a result, the majority of the thrust maneuver at lower thrust-to-mass ratios is the first thrust which is followed by a short coast and a minimal second thrust for final orbit corrections.

Chapter 5

Conclusion and Recommendations for Future Work

Atmospheric drag at low altitude orbits can dramatically affect the longevity of a spacecraft. As a result of low altitude satellites usefulness, it is imperative that an optimal re-boost trajectory is determined to increase the lifetime of the spacecraft. Particle Swarm Optimization is used to find the optimal solution and is successful in most cases. On occasion, a local minimum is produced as the final solution so results can vary as much as 20%. Three re-boost methods are tested which included a single-continuous thrust maneuver, a two thrust maneuver, and a variable thrust maneuver. The two thrust maneuver is determined to be the best method because it used the least amount of fuel and always meet the required final conditions. Using the two thrust method, other spacecraft parameters are varied which include, starting and minimum altitudes, the coefficient of drag, the spacecraft's cross-sectional diameter, the spacecraft's mass, the effective exhaust velocity of the thruster, and the thrust-to-mass ratio of the spacecraft.

In the future, each test should be run with the single-continuous thrust method and the variable thrust method which should give incite to why those methods are especially error prone. Also, the tests should be rerun using more particles and more iterations to see if the program could find the global minimum more consistently in the first attempt. Additionally, it should be noted that rerunning the same test several times and taking the best result will most likely give more accurate data. Overall, particle swarm optimization is effectively used in solving this optimization problem and has the potential to find better results with some slight modifications and more testing.

References

[1] Pontani, Mauro, and Bruce A. Conway. *Particle Swarm Optimization Applied to Impulsive Orbital Transfers*. University of Illinois at Urbana-Champaign, 2011. Print.

[2] Pontani, Mauro, and Bruce A. Conway. *Particle Swarm Optimization Applied to Space Trajectories*. University of Illinois at Urbana-Champaign, 2011. Print.

[3] Zhu, Kai-Jian, Jun-Feng Li, and He-Xi Baoyin. *Satellite Scheduling Considering Maximum Observation Coverage Time and Minimum Orbital Transfer Fuel Cost*. Tsinghua University, Beijing, China, 2009. Print.

[4] Fujita, Kazuhisa, and Atsushi Noda. *Rarefied Aerodynamics of a Super Low Altitude Test Satellite*. Japan Aerospace Exploration Agency, Chofu, Tokyo, 2009. Print.

[5] Doornbos, E., H. Klinkrad, and P. Visser. *Atmospheric Density Calibration Using Satellite Drag Observations*. Delft University of Technology, The Netherlands. Print.

[6] "Properties of Standard Atmosphere." *Www.braeunig.us*. Braeunig. Web. 20 Feb. 2012. <http://www.braeunig.us/space/atmos.htm>.

[7] Curtis, Howard D. *Orbital Mechanics for Engineering Students*. 2nd ed. Amsterdam: Butterworth-Heinemann, 2010. Print.

Appendix A

Atmospheric Data

Table A1-1 below shows the atmospheric density of air at the altitudes from 140 km – 500 km which is the range of altitudes the code can accurately measure spaceflight dynamics. The data is taken from the MSISE-90 Model of the Earth's Upper Atmosphere[6].

Table A1-1 Air Density from MSISE-90 Model

Altitude (km)	Density (kg/m3)
140	3.26E-09
160	1.18E-09
180	5.51E-10
200	2.91E-10
220	1.66E-10
240	9.91E-11
260	6.16E-11
280	3.94E-11
300	2.58E-11
320	1.72E-11
340	1.16E-11
360	7.99E-12
380	5.55E-12
400	3.89E-12
420	2.75E-12
440	1.96E-12
460	1.40E-12
480	1.01E-12
500	7.30E-13

Using a 100 km step size, a series of density vs. altitude plots determined a series of line of best fit equations to model the atmospheric density between the data points in Table A1-1. The plots, including their lines of best fit and their R^2 values are shown below in Figure A1-1.

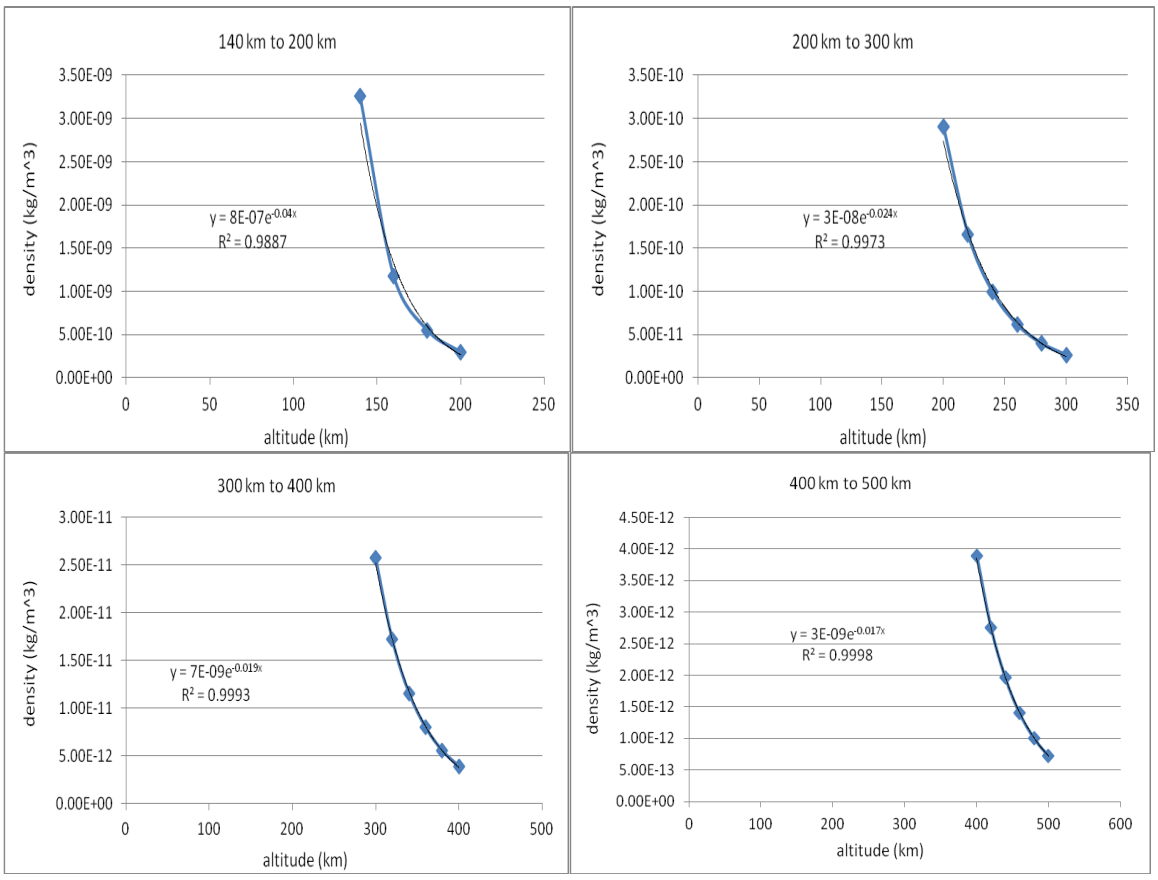


Figure A1-1 Altitude vs. Density Plots

Appendix B

Single Thrust Source Code

The following code is the code used to run the testing for the single-continuous thrust case. It contains a main code, 6 functions, and 2 ode45 functions.

PSOTest_adaptive (Main)

```
%% PSO With Variable Accelerator Coefficients

clc;
clear all;

%define PSO variables and arrays
global J JBest
global GG GBest GGstar
global N_particles N_elements N_iterations
global P PBest V
global BLv BUv BLp BUp

%define
global t1
global mass_ratio

%define particles and elements
N_particles = 50;
N_elements = 7;
N_iterations = 200;

%define s/c and orbit conditions
global muB R1 R2
global c n0 Mass Cd A

%initialize s/c and orbit conditions
t1=15000; %t1 in TU (should be adjusted to get faster run-times)
muB = 1;  %in DU^3/TU^2
c = .5;  %effective exhaust velocity
n0 = .05;  %Thrust to mass ratio at t0
Mass=50000;  %Mass of s/c in kg
Cd=2.5; %coefficient of drag
```

```
D=5/(6378.1*1000); %diameter of the s/c cross section in DU
A=pi*D^2/4; %Cross-sectional area of the s/c in DU^2
R1 = (200+6378.1)/6378.1; %min radius of s/c (alt + radius of Earth) in
DU
R2 = (250+6378.1)/6378.1; %max radius of s/c (alt + radius of Earth) in
DU

%% create random initial population
P = rand(N_particles,N_elements);

% Restricting to boundaries
for i=1:N_elements
    if i<7
        P(:,i) = -1+2*P(:,i);
    elseif i==7
        P(:,i) = (3-1e-3)*P(:,i)+1e-3;
    end
end

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

%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1 -1 -1 -1 -1 -1 1e-3];
BUp = [1 1 1 1 1 1 3];
%1-6 are the thrust angles
%7 is the thrust time

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

%% Solve

Decay;

for j = 1:N_iterations
    fprintf('Iteration number: %g \n', j)
    EvalJ;
    EvalPGBest;
    UpdateV(j);
    UpdateP;
    GGstar(j) = GG;
    fprintf('GG = %6.5f \n',GG)
    fprintf('dt1 = %6.5f, dt2 = %6.5f \n',t1, GBest(7))
    fprintf('mf/m0 = %6.5f \n', mass_ratio)
```

```
end

%% Plots and Final Results

FinalResults;
```

Decay

```
function Decay()
%% Decay function

global t1 t1_save R1 rm x_decay x_save
global muB R1 R2 %#ok<REDEF>

%define initial orbit
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

    x0 = [r0; v_r0; theta0; theta_dot0];
    tspan = [0  t1];
    options=odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
    [t,x] = ode45('PSOode', tspan, x0,options);
    endlimit = length(x);
    rm=min(x(:,1));

    for i=1:1:endlimit
        x_decay(i,:)=x(i,:);
        if abs(R1-x(i,1))<=1e-5
            t1_save=t(i);
            x_save=x(i,:);
            break;
        end
    end

    t1=t1_save;

end
```

EvalJ

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

global P J N_particles
global t1 t2 x_save rm
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5

%% Initialize values
global muB R2


%% Start calculations to calculate J
for i = 1:N_particles
    zeta0 = P(i,1);
    zeta1 = P(i,2);
    zeta2 = P(i,3);
    zeta3 = P(i,4);
    zeta4 = P(i,5);
    zeta5 = P(i,6);
    t2 = P(i,7);

    %begin thrust to orginal radius
    x2 = [x_save(1); x_save(2); x_save(3); x_save(4)];
    tf = t2 + t1;

    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit = length(x);

    if rm > min(x(:,1))
        rm = min(x(:,1));
    end

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

    v_thetaf = rf*theta_dotf;

    %Calculate J
    d = zeros(3,1);
```

```
    d(1) = v_rf*2;
    d(2) = (v_thetaf - sqrt(muB/R2));
    d(3) = (rf - R2)*25;

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

    J(i) = t2 + sum(K);

end
```

EvalPGBest

```
function EvalPGBest()
%% EvalP&GBest determines the best position visited by particle i up
%% through N_Particles
%%   the current iteration)

global P J JBest PBest GG GBest N_particles best_particle
global mass_ratio c n0

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

for i = 1:N_particles
    if J(i) < GG
       GG = J(i);
       GBest = P(i,:);
       mass_ratio = 1 - (n0/c)*(GBest(7));
       best_particle=i;
    end

end
```

UpdateV

```
function UpdateV(j) % UpdateV updates the velocity vector V

% Variable accelerator coeffs.
global P PBest GBest N_particles N_elements V BLv BUv

%% RGM set c_C and c_S to same form as Pontani & Conway

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

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

UpdateP

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

global P N_particles N_elements V BLp BUp

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

FinalResults

```
function FinalResults()
%% Prints the final results for the best particle

global P best_particle GGstar N_iterations
global t1 t2 mass_ratio x_decay
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5
global muB R2
global Mass

    %% Initialize values
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

    %% Start calculations to calculate optimum solution
    zeta0 = P(best_particle,1);
    zeta1 = P(best_particle,2);
    zeta2 = P(best_particle,3);
    zeta3 = P(best_particle,4);
    zeta4 = P(best_particle,5);
    zeta5 = P(best_particle,6);
    t2 = P(best_particle,7);

    endlimit1 = length(x_decay);
    rm=min(x_decay(:,1));
    decay_theta=x_decay(endlimit1,3);
    decay_r=x_decay(:,1)*6378.1;
    decay_t=x_decay(:,3);

    %begin thrust to orginal radius
    x2 = [x_decay(endlimit1,1); x_decay(endlimit1,2);
x_decay(endlimit1,3); x_decay(endlimit1,4)];
    tf = t2 + t1;

    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit2 = length(x);
    thrust_r=x(:,1)*6378.1;
    thrust_t=x(:,3);
```

```
    if rm > min(x(:,1))
        rm = min(x(:,1));
    end

    rf = x(endlimit2,1);
    v_rf = x(endlimit2,2);
    theta_f = x(endlimit2,3);
    theta_dotf = x(endlimit2,4);
    thrust_theta = x(endlimit2,3)-decay_theta;

    v_thetaf = rf*theta_dotf;

    %Errors
    d = zeros(4,1);
    d(1) = v_rf;
    d(2) = v_thetaf - sqrt(muB/R2);
    d(3) = (rf - R2)*100;

    %Conversions to metric units
    rm=(rm-1)*6378.1;  %minimum altitude (km)
    rf=(rf-1)*6378.1;  %final altitude (km)
    v_rf=v_rf*6378.1/806.80455; %final radial velocity in (km/s)
    d(2)=d(2)*6378.1/806.80455; %final difference in transverse
velocity (km/s)
    N_rev_decay=decay_theta/(2*pi);
    N_rev_thrust=thrust_theta/(2*pi);
    Fuel_used=Mass*(1-mass_ratio); %fuel used in kg
    Fuel_per_Rev=Fuel_used/(N_rev_decay+N_rev_thrust); %in kg/rev
    tf=tf*(806.80455)/86400; %cycle time in days
    t1=t1*(806.80455)/86400; %decay time in days
    t2=t2*(806.80455)/60; %thrust time in minutes
    decay_alt=(decay_r-6378.1); %decay altitude
    elements=length(decay_r); %length of decay
    time_decay=[0:t1/(elements-1):t1]; %time array for decay
    thrust_alt=(thrust_r-6378.1); %thrust altitude
    elements=length(thrust_r); %length of thrust
    time_thrust=[0:t2/(elements-1):t2]; %time array for thrust


    %plots
    iterations = [1:N_iterations];
    subplot(3,2,1)
    hold on
    plot(iterations, log(GGstar))
    title('GGstar vs. iteration number')
    xlabel('iteration')
    ylabel('log(GGstar)')

    subplot(3,2,2)
    polar(decay_t,decay_alt,'r')
    title('Altitude vs. theta - Decay Section')
```

```
    subplot(3,2,3);
    polar(thrust_t,thrust_alt,'b')
    title('Altitude vs. theta - Thrust Section')

    subplot(3,2,4)
    plot(time_decay,decay_alt)
    title('Altitude (km) vs. time (days) - Decay Section')
    xlabel('time (days)')
    ylabel('Altitude (km)')

    subplot(3,2,5)
    plot(time_thrust,thrust_alt)
    title('Altitude (km) vs. time (minutes) - Thrust Section')
    xlabel('time (minutes)')
    ylabel('Altitude (km)')

    save 'trajectory'

    %Print Statements
    fprintf('\n')
    fprintf('Final Results:')
    fprintf('Minimum Altitude = %6.5f (km) \n',rm)
    fprintf('Final Altitude = %6.5f (km) \n',rf)
    fprintf('Final Radial Velocity = %6.5f (km/s) \n',v_rf)
    fprintf('Final Difference in Transverse Velocity = %6.5f (km/s)
\n',d(2))
    fprintf('Revolutions of Decay = %6.5f \n',N_rev_decay)
    fprintf('Revolutions of Thrust = %6.5f \n',N_rev_thrust)
    fprintf('Fuel Used = %6.5f (kg) \n',Fuel_used)
    fprintf('Fuel Used per Revolution = %6.5f (kg/Rev)
\n',Fuel_per_Rev)
    fprintf('Time per Cycle = %6.5f (days) \n',tf)
    fprintf('Time of Decay = %6.5f (days) \n',t1)
    fprintf('Time of Thrust = %6.5f (minutes) \n',t2)

end
```

PSOode

```
function [ xdot ] = PSOode( t,x )
%% State variables for PSO algorithm for decay
% 0 < t < t1

xdot = zeros(4,1);

%define s/c and orbit conditions
```

```
global muB
global Mass Cd A

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
end

%change density and Mass to canonical units
conv = density/Mass * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1);

end
```

PSOode2

```
function [ xdot ] = PSOode2( t,x )
%% State variables for PSO algorithm for thrust arc
% t1 < t < tf

global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5
global muB
global c n0 Mass Cd A

xdot = zeros(4,1);

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
```

```
end

%change density and Mass to canonical units
conv = density/(Mass*(1-n0*t)) * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2) + ((c*n0)/(c -
n0*t))*sin(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1) + ((c*n0)/(c -
n0*t))*cos(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5)/x(1);

end
```

Appendix C

Two Thrust Source Code

The following code is the code used to run the testing for the two thrust case. It contains 7 of the 9 required functions: a main code, 4 functions, and 2 ode45 functions. In addition, the UpdateV and UpdateP m-files from the single-continuous thrust case should be included to complete the algorithm.

PSOTest_adaptive (Main)

```
%% PSO With Variable Accelerator Coefficients

clc;
clear all;

%define PSO variables and arrays
global J JBest
global GG GBest GGstar
global N_particles N_elements N_iterations
global P PBest V
global BLv BUv BLp BUp

%define
global t1
global mass_ratio

%define particles and elements
N_particles = 50;
N_elements = 15;
N_iterations = 250;

%define s/c and orbit conditions
global muB R1 R2
global c n0 Mass Cd A

%initialize s/c and orbit conditions
muB = 1;  %in DU^3/TU^2
```

```
c = 0.5;  %effective exhaust velocity
n0 = .05;  %Thrust to mass ratio at t0
Mass=50000;  %Mass of s/c in kg
Cd=2.5; %coefficient of drag
D=5/(6378.1*1000); %diameter of the s/c cross section in DU
A=pi*D^2/4; %Cross-sectional area of the s/c
R1 = (200+6378.1)/6378.1; %min radius of s/c (alt + radius of Earth) in
DU
R2 = (250+6378.1)/6378.1; %max radius of s/c (alt + radius of Earth) in
DU

%% create random initial population
P = rand(N_particles,N_elements);

% Restricting to boundaries
for i=1:N_elements
    if i<13
        P(:,i) = -1+2*P(:,i);
    elseif i==14
        P(:,i) = (5-1e-3)*P(:,i)+1e-3;
    else
        P(:,i) = (1.0-1e-4)*P(:,i)+1e-4;
    end
end

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

%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1e-4 1e-3 1e-4];
BUp = [1 1 1 1 1 1 1 1 1 1 1 1 1.0 5 1.0];
%1-6 are the thrust angles for first thrust
%7-12 are the thrust angles for second thrust
%13 is the thrust time for first thrust
%14 is the time between thrusts
%15 is the thrust time for the second thrust

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

%% Solve

Decay;

for j = 1:N_iterations
```

```
    fprintf('Iteration number: %g \n', j)
    EvalJ;
    EvalPGBest;
    UpdateV(j);
    UpdateP;
    GGstar(j) = GG;
    fprintf('GG = %6.5f \n',GG)
    fprintf('dt1 = %6.5f, dt2 = %6.5f, dt3 = %6.5f, dt4 = %6.5f \n',t1,
GBest(13), GBest(14), GBest(15))
    fprintf('mf/m0 = %6.5f \n', mass_ratio)
end

%% Plots and Final Results

FinalResults;
```

Decay

```
function Decay()
%% Decay function

global t1 rm x_decay x_save Mass_hold Mass
global muB R1 R2    %#ok<REDEF>

%define initial orbit
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

    t1=11000;
    Mass_hold=Mass;
    x0 = [r0; v_r0; theta0; theta_dot0];
    tspan = [0  t1];
    options=odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
    [t,x] = ode45('PSOode', tspan, x0,options);
    endlimit = length(x);
    rm=x(endlimit,1);

    for i=1:1:endlimit
        x_decay(i,:)=x(i,:);
        if abs(R1-x(i,1))<=1e-6
            t1_save=t(i);
            x_save=x(i,:);
            rm=x(i,1);
            break;
```

```
        end
    end

    t1=t1_save;

end
```

EvalJ

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

global P J N_particles
global t1 t2 x_save rm Mass Mass_hold n0 c
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5

%% Initialize values
global muB R2


%% Start calculations to calculate J
for i = 1:N_particles
    zeta0 = P(i,1);
    zeta1 = P(i,2);
    zeta2 = P(i,3);
    zeta3 = P(i,4);
    zeta4 = P(i,5);
    zeta5 = P(i,6);
    t2 = P(i,13);
    t3 = P(i,14);
    t4 = P(i,15);
    tf = t1 + t2 + t3 + t4;

    %begin first thrust to coast arc

    Mass_hold=Mass;

    x2 = [x_save(1); x_save(2); x_save(3); x_save(4)];

    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit = length(x);

    if rm > min(x(:,1))
        rm = min(x(:,1));
    end
```

```
Mass_hold=Mass*(1-(n0/c)*t2);

%begin coast arc
x3 = [x(endlimit,1); x(endlimit,2); x(endlimit,3); x(endlimit,4)];

tspan3 = [0 t3];
options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
[t,x] = ode45('PSOode',tspan3,x3,options);
endlimit = length(x);

if rm > min(x(:,1))
    rm = min(x(:,1));
end

%begin second thrsut to original radius
zeta0 = P(i,7);
zeta1 = P(i,8);
zeta2 = P(i,9);
zeta3 = P(i,10);
zeta4 = P(i,11);
zeta5 = P(i,12);

x4 = [x(endlimit,1); x(endlimit,2); x(endlimit,3); x(endlimit,4)];

tspan4 = [0 t4];
options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
[t,x] = ode45('PSOode2',tspan4,x4,options);
endlimit = length(x);

if rm > min(x(:,1))
    rm = min(x(:,1));
end

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

v_thetaf = rf*theta_dotf;

%Calculate J
d = zeros(4,1);
d(1) = v_rf*2;
d(2) = (v_thetaf - sqrt(muB/R2))*1.2;
d(3) = (rf - R2)*20;

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

```
        else
            alpha(j) = 100;
        end
        K(j) = abs(d(j))*alpha(j);
    end

    J(i) = t2 + t4 + sum(K);

end
```

EvalPGBest

```
function EvalPGBest()
%% EvalP&GBest determines the best position visited by particle i up
through
%%   the current iteration)

global P J JBest PBest GG GBest N_particles best_particle
global mass_ratio c n0

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

for i = 1:N_particles
    if J(i) < GG
       GG = J(i);
       GBest = P(i,:);
       mass_ratio = 1 - (n0/c)*(GBest(13)+GBest(15));
       best_particle=i;
    end

end
```

FinalResults

```
function FinalResults()
%% Prints the final results for the best particle

global P best_particle GGstar N_iterations
global t1 t2 t3 t4 mass_ratio x_decay
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5
global muB R2
global Mass

%% Initialize values
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

%% Start calculations to calculate optimum solution
    zeta0 = P(best_particle,1);
    zeta1 = P(best_particle,2);
    zeta2 = P(best_particle,3);
    zeta3 = P(best_particle,4);
    zeta4 = P(best_particle,5);
    zeta5 = P(best_particle,6);
    t2 = P(best_particle,13);
    t3 = P(best_particle,14);
    t4 = P(best_particle,15);
    tf = t1 + t2 + t3 + t4;

    %decay solutions
    endlimit1 = length(x_decay);
    rm=min(x_decay(:,1));
    decay_theta=x_decay(endlimit1,3);
    decay_r=x_decay(:,1)*6378.1;
    decay_t=x_decay(:,3);

    %begin first thrust to coast arc
    x2 = [x_decay(endlimit1,1); x_decay(endlimit1,2);
x_decay(endlimit1,3); x_decay(endlimit1,4)];

    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit = length(x);
```

```
thrust_r1=x(:,1)*6378.1;
thrust_t1=x(:,3);

if rm > min(x(:,1))
    rm = min(x(:,1));
end

%begin coast arc
x3 = [x(endlimit,1); x(endlimit,2); x(endlimit,3); x(endlimit,4)];

tspan3 = [0 t3];
options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
[t,x] = ode45('PSOode',tspan3,x3,options);
endlimit = length(x);
coast_r=x(:,1)*6378.1;
coast_t=x(:,3);

if rm > min(x(:,1))
    rm = min(x(:,1));
end

%begin second thrsut to original radius
zeta0 = P(best_particle,7);
zeta1 = P(best_particle,8);
zeta2 = P(best_particle,9);
zeta3 = P(best_particle,10);
zeta4 = P(best_particle,11);
zeta5 = P(best_particle,12);

x4 = [x(endlimit,1); x(endlimit,2); x(endlimit,3); x(endlimit,4)];

tspan4 = [0 t4];
options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
[t,x] = ode45('PSOode2',tspan4,x4,options);
endlimit = length(x);
thrust_r2=x(:,1)*6378.1;
thrust_t2=x(:,3);

if rm > min(x(:,1))
    rm = min(x(:,1));
end

rf = x(endlimit,1);
v_rf = x(endlimit,2);
theta_f = x(endlimit,3);
theta_dotf = x(endlimit,4);
thrust_theta = x(endlimit,3)-decay_theta;

v_thetaf = rf*theta_dotf;

%Errors
```

```
    d = zeros(4,1);
    d(1) = v_rf;
    d(2) = v_thetaf - sqrt(muB/R2);
    d(3) = (rf - R2)*100;

    %Conversions to metric units
    rm=(rm-1)*6378.1;  %minimum altitude (km)
    rf=(rf-1)*6378.1;  %final altitude (km)
    v_rf=v_rf*6378.1/806.80455; %final radial velocity in (km/s)
    d(2)=d(2)*6378.1/806.80455; %final difference in transverse
velocity (km/s)
    N_rev_decay=decay_theta/(2*pi);
    N_rev_thrust=thrust_theta/(2*pi); %revolutions of the whole
maneuver
    Fuel_used=Mass*(1-mass_ratio); %fuel used in kg
    Fuel_per_Rev=Fuel_used/(N_rev_decay+N_rev_thrust); %in kg/rev
    tf=tf*(806.80455)/86400; %cycle time in days
    t1=t1*(806.80455)/86400; %decay time in days
    t2=t2*(806.80455)/60; %thrust1 time in minutes
    t3=t3*(806.80455)/60; %coast time in minutes
    t4=t4*(806.80455)/60; %thrust2 time in minutes

    decay_alt=(decay_r-6378.1); %decay altitude
    elements=length(decay_r); %length of decay
    time_decay=[0:t1/(elements-1):t1]; %time array for decay
    thrust1_alt=(thrust_r1-6378.1); %thrust1 altitude
    elements=length(thrust_r1); %length of thrust1
    time_thrust1=[0:t2/(elements-1):t2]; %time array for thrust1
    coast_alt=(coast_r-6378.1); %coast altitude
    elements=length(coast_r); %length of coast
    time_coast=[t2:t3/(elements-1):t3+t2]; %time array for coast
    thrust2_alt=(thrust_r2-6378.1); %thrust2 altitude
    elements=length(thrust_r2); %length of thrust2
    time_thrust2=[t2+t3:t4/(elements-1):t2+t3+t4]; %time array for
thrust2


    %plots
    iterations = [1:N_iterations];
    subplot(3,2,1)
    hold on
    plot(iterations, log(GGstar))
    title('GGstar vs. iteration number')
    xlabel('iteration')
    ylabel('log(GGstar)')

    subplot(3,2,2)
    polar(decay_t,decay_alt,'b')
    title('Altitude vs. theta - Decay Section')

    subplot(3,2,3);
    polar(thrust_t1,thrust1_alt,'b')
```

```
    hold on;
    polar(coast_t,coast_alt,'r')
    polar(thrust_t2,thrust2_alt,'g')
    title('Altitude vs. theta - Thrust Maneuver Section')

    subplot(3,2,4)
    plot(time_decay,decay_alt)
    title('Altitude (km) vs. time (days) - Decay Section')
    xlabel('time (days)')
    ylabel('Altitude (km)')

    subplot(3,2,5)
    plot(time_thrust1,thrust1_alt,'b')
    hold on;
    plot(time_coast,coast_alt,'r')
    plot(time_thrust2,thrust2_alt,'g')
    title('Altitude (km) vs. time (minutes) - Thrust Maneuver Section')
    xlabel('time (minutes)')
    ylabel('Altitude (km)')

    save 'trajectory'

    %Print Statements
    fprintf('\n')
    fprintf('Final Results:')
    fprintf('Minimum Altitude = %6.5f (km) \n',rm)
    fprintf('Final Altitude = %6.5f (km) \n',rf)
    fprintf('Final Radial Velocity = %6.5f (km/s) \n',v_rf)
    fprintf('Final Difference in Transverse Velocity = %6.5f (km/s)
\n',d(2))
    fprintf('Revolutions of Decay = %6.5f \n',N_rev_decay)
    fprintf('Revolutions of Thrust Maneuver (including coast)= %6.5f
\n',N_rev_thrust)
    fprintf('Fuel Used = %6.5f (kg) \n',Fuel_used)
    fprintf('Fuel Used per Revolution = %6.5f (kg/Rev)
\n',Fuel_per_Rev)
    fprintf('Time per Cycle = %6.5f (days) \n',tf)
    fprintf('Time of Decay = %6.5f (days) \n',t1)
    fprintf('Time of Thrust1 = %6.5f (minutes) \n',t2)
    fprintf('Time of Coast = %6.5f (minutes) \n',t3)
    fprintf('Time of Thrust2 = %6.5f (minutes) \n',t4)

end
```

PSOode

```
function [ xdot ] = PSOode( t,x )
%% State variables for PSO algorithm for decay
% 0 < t < t1 and 0 < t < t3

xdot = zeros(4,1);

%define s/c and orbit conditions
global muB
global Mass_hold Cd A

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
end

%change density and Mass to canonical units
conv = density/Mass_hold * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1);

end
```

PSOode2

```
function [ xdot ] = PSOode2( t,x )
%% State variables for PSO algorithm for thrust arc
% 0 < t < t2 and 0 < t < t4

global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5
global muB
global c n0 Mass_hold Cd A

xdot = zeros(4,1);

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
end

%change density and Mass to canonical units
conv = density/(Mass_hold*(1-n0*t)) * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2) + ((c*n0)/(c -
n0*t))*sin(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1) + ((c*n0)/(c -
n0*t))*cos(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5)/x(1);

end
```

Appendix D

Variable Thrust Magnitude Source Code

The following code is the code used to run the testing for the variable thrust case. It contains 7 of the 9 required functions: a main code, 4 functions, and 2 ode45 functions. In addition, the UpdateV and UpdateP m-files from the single-continuous thrust case should be included to complete the algorithm.

PSOTest_adaptive (Main)

```
%% PSO With Variable Accelerator Coefficients

clc;
clear all;

%define PSO variables and arrays
global J JBest
global GG GBest GGstar
global N_particles N_elements N_iterations
global P PBest V
global BLv BUv BLp BUp

%define
global t1
global mass_ratio

%define particles and elements
N_particles = 50;
N_elements = 13;
N_iterations = 200;

%define s/c and orbit conditions
global muB R1 R2
global c n0 Mass Cd A

%initialize s/c and orbit conditions
muB = 1;  %in DU^3/TU^2
c = .5;  %effective exhaust velocity
n0 = .05;  %Thrust to mass ratio at t0
Mass=50000;  %Mass of s/c in kg
Cd=2.5; %coefficient of drag
```

```
D=5/(6378.1*1000); %diameter of the s/c cross section in DU
A=pi*D^2/4; %Cross-sectional area of the s/c
R1 = (200+6378.1)/6378.1; %min radius of s/c (alt + radius of Earth) in
DU
R2 = (250+6378.1)/6378.1; %max radius of s/c (alt + radius of Earth) in
DU

%% create random initial population
P = rand(N_particles,N_elements);

% Restricting to boundaries
for i=1:N_elements
    if i<7
        P(:,i) = -1+2*P(:,i);
    elseif i==7
        P(:,i) = (3-1e-3)*P(:,i)+1e-3;
    elseif i==8
        P(:,i) = P(:,i);
    elseif i>8
        P(:,i) = -1+2*P(:,i);
    end
end

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

%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1 -1 -1 -1 -1 -1 1e-3 0 -1 -1 -1 -1 -1];
BUp = [1 1 1 1 1 1 3 1 1 1 1 1 1 ];
%1-6 are the thrust angles
%7 is the thrust time
%8-13 is the thrust magnitude

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

%% Solve

Decay;

for j = 1:N_iterations
    fprintf('Iteration number: %g \n', j)
    EvalJ;
    EvalPGBest;
    UpdateV(j);
```

```
    UpdateP;
    GGstar(j) = GG;
    fprintf('GG = %6.5f \n',GG)
    fprintf('dt1 = %6.5f, dt2 = %6.5f \n',t1, GBest(7))
    fprintf('mf/m0 = %6.5f \n', mass_ratio)
end

%% Plots and Final Results

FinalResults;
```

Decay

```
function Decay()
%% Decay function

global t1 t1_save R1 rm x_decay x_save
global muB R1 R2 %#ok<REDEF>

%define initial orbit
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

    t1=15000;
    x0 = [r0; v_r0; theta0; theta_dot0];
    tspan = [0  t1];
    options=odeset('RelTol', 1e-8, 'AbsTol', 1e-8);
    [t,x] = ode45('PSOode', tspan, x0,options);
    endlimit = length(x)
    rm=min(x(:,1));

    for i=1:1:endlimit
        x_decay(i,:)=x(i,:);
        if abs(R1-x(i,1))<=1e-5
            t1_save=t(i);
            x_save=x(i,:);
            break;
        end
    end

    t1=t1_save;

end
```

EvalJ

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

global P J N_particles
global t1 t2 x_save rm v_Mass tlast integration Mass n0
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5 beta0 beta1 beta2 beta3
beta4 beta5

%% Initialize values
global muB R2


%% Start calculations to calculate J
for i = 1:N_particles
    zeta0 = P(i,1);
    zeta1 = P(i,2);
    zeta2 = P(i,3);
    zeta3 = P(i,4);
    zeta4 = P(i,5);
    zeta5 = P(i,6);
    t2 = P(i,7);
    beta0 = P(i,8);
    beta1 = P(i,9);
    beta2 = P(i,10);
    beta3 = P(i,11);
    beta4 = P(i,12);
    beta5 = P(i,13);

    %begin thrust to orginal radius
    x2 = [x_save(1); x_save(2); x_save(3); x_save(4)];
    tf = t2 + t1;

    v_Mass=Mass;
    tlast=0;
    integration=0;
    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit = length(x);

    if rm > min(x(:,1))
        rm = min(x(:,1));
    end

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

```
    v_thetaf = rf*theta_dotf;

    %Analyze thrust function
    for k=1:1001
        j(k) = t2/1000 * (k-1);
        n_t(k) = beta0 + beta1*j(k) + beta2*j(k)^2 + beta3*j(k)^3 +
beta4*j(k)^4 + beta5*j(k)^5;
    end

    n_t_avg=abs(trapz(j,n_t)/t2);
    top=max(n_t);
    bottom=min(n_t);

    %Calculate J
    d = zeros(4,1);
    d(1) = v_rf*2;
    d(2) = (v_thetaf - sqrt(muB/R2));
    d(3) = (rf - R2)*15;
    d(4) = top-1;
    d(5) = bottom;

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

    if d(4)>0
        alpha(4)=100;
    else
        alpha(4)=0;
    end

    if d(5)<0
        alpha(5)=100;
    else
        alpha(5)=0;
    end

    for j=1:5
        K(j) = abs(d(j))*alpha(j);
    end

    J(i) = t2*n_t_avg + sum(K);

end
```

EvalPGBest

```
function EvalPGBest()
%% EvalP&GBest determines the best position visited by particle i up
through
%%   the current iteration)

global P J JBest PBest GG GBest N_particles best_particle
global mass_ratio c n0 t2
for i = 1:N_particles
    if J(i) < JBest(i)
        PBest(i,:) = P(i,:);
        JBest(i) = J(i);
    end
end

for i = 1:N_particles
    if J(i) < GG
       GG = J(i);
       GBest = P(i,:);

        for k=1:1001
            j(k) = t2/1000 * (k-1);
            n(k)=n0*(P(i,8) + P(i,9)*j(k) + P(i,10)*j(k)^2 +
P(i,11)*j(k)^3 + P(i,12)*j(k)^4 + P(i,13)*j(k)^5);
        end

       n_avg=abs(trapz(j,n)/t2);
       mass_ratio = 1 - (n_avg/c)*(GBest(7));
       best_particle=i;
    end

end
```

FinalResults

```
function FinalResults()
%% Prints the final results for the best particle

global P best_particle GGstar N_iterations
global t1 t2 mass_ratio x_decay n0
global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5 beta0 beta1 beta2 beta3
beta4 beta5
global muB R2
global Mass
```

```
    %% Initialize values
    v_r0 = 0;
    v_theta0 = sqrt(muB/R2);
    r0 = R2;
    xi0 = 0;
    theta0 = 0;
    theta_dot0 = v_theta0/r0;

    %% Start calculations to calculate optimum solution
    zeta0 = P(best_particle,1);
    zeta1 = P(best_particle,2);
    zeta2 = P(best_particle,3);
    zeta3 = P(best_particle,4);
    zeta4 = P(best_particle,5);
    zeta5 = P(best_particle,6);
    t2 = P(best_particle,7);
    beta0 = P(best_particle,8);
    beta1 = P(best_particle,9);
    beta2 = P(best_particle,10);
    beta3 = P(best_particle,11);
    beta4 = P(best_particle,12);
    beta5 = P(best_particle,13);

    endlimit1 = length(x_decay);
    rm=min(x_decay(:,1));
    decay_theta=x_decay(endlimit1,3);
    decay_r=x_decay(:,1)*6378.1;
    decay_t=x_decay(:,3);

    %begin thrust to orginal radius
    x2 = [x_decay(endlimit1,1); x_decay(endlimit1,2);
x_decay(endlimit1,3); x_decay(endlimit1,4)];
    tf = t2 + t1;

    tspan2 = [0 t2];
    options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
    [t,x] = ode45('PSOode2',tspan2,x2,options);
    endlimit2 = length(x);
    thrust_r=x(:,1)*6378.1;
    thrust_t=x(:,3);

    if rm > min(x(:,1))
        rm = min(x(:,1));
    end

    rf = x(endlimit2,1);
    v_rf = x(endlimit2,2);
    theta_f = x(endlimit2,3);
    theta_dotf = x(endlimit2,4);
    thrust_theta = x(endlimit2,3)-decay_theta;
```

```
    v_thetaf = rf*theta_dotf;

    %Errors
    d = zeros(4,1);
    d(1) = v_rf;
    d(2) = v_thetaf - sqrt(muB/R2);
    d(3) = (rf - R2)*100;

    %Conversions to metric units
    rm=(rm-1)*6378.1;  %minimum altitude (km)
    rf=(rf-1)*6378.1;  %final altitude (km)
    v_rf=v_rf*6378.1/806.80455; %final radial velocity in (km/s)
    d(2)=d(2)*6378.1/806.80455; %final difference in transverse
velocity (km/s)
    N_rev_decay=decay_theta/(2*pi);
    N_rev_thrust=thrust_theta/(2*pi);
    Fuel_used=Mass*(1-mass_ratio); %fuel used in kg
    Fuel_per_Rev=Fuel_used/(N_rev_decay+N_rev_thrust); %in kg/rev
    tf=tf*(806.80455)/86400; %cycle time in days
    t1=t1*(806.80455)/86400; %decay time in days
    t2=t2*(806.80455)/60; %thrust time in minutes
    decay_alt=(decay_r-6378.1); %decay altitude
    elements=length(decay_r); %length of decay
    time_decay=[0:t1/(elements-1):t1]; %time array for decay
    thrust_alt=(thrust_r-6378.1); %thrust altitude
    elements=length(thrust_r); %length of thrust
    time_thrust=[0:t2/(elements-1):t2]; %time array for thrust

    %Analyze thrust function
    for k=1:1001
        j(k) = t2/1000 * (k-1) *(60/806.80455);
        n_t(k) = beta0 + beta1*j(k) + beta2*j(k)^2 + beta3*j(k)^3 +
beta4*j(k)^4 + beta5*j(k)^5;
    end

    %plots
    iterations = [1:N_iterations];
    subplot(3,2,1)
    hold on
    plot(iterations, log(GGstar))
    title('GGstar vs. iteration number')
    xlabel('iteration')
    ylabel('log(GGstar)')

    subplot(3,2,2)
    polar(decay_t,decay_alt,'r')
    title('Altitude vs. theta - Decay Section')

    subplot(3,2,3);
    polar(thrust_t,thrust_alt,'b')
```

```
    title('Altitude vs. theta - Thrust Section')

    subplot(3,2,4)
    plot(time_decay,decay_alt)
    title('Altitude (km) vs. time (days) - Decay Section')
    xlabel('time (days)')
    ylabel('Altitude (km)')

    subplot(3,2,5)
    plot(time_thrust,thrust_alt)
    title('Altitude (km) vs. time (minutes) - Thrust Section')
    xlabel('time (minutes)')
    ylabel('Altitude (km)')

    subplot(3,2,6)
    plot(j,n_t)
    title('Thrust-to-Mass Ratio vs. Time')
    xlabel('time (TU)')
    ylabel('Thrust-to-Mass Ratio(DU/TU^2)')

    save 'trajectory'

    %Print Statements
    fprintf('\n')
    fprintf('Final Results:')
    fprintf('Minimum Altitude = %6.5f (km) \n',rm)
    fprintf('Final Altitude = %6.5f (km) \n',rf)
    fprintf('Final Radial Velocity = %6.5f (km/s) \n',v_rf)
    fprintf('Final Difference in Transverse Velocity = %6.5f (km/s)
\n',d(2))
    fprintf('Revolutions of Decay = %6.5f \n',N_rev_decay)
    fprintf('Revolutions of Thrust = %6.5f \n',N_rev_thrust)
    fprintf('Fuel Used = %6.5f (kg) \n',Fuel_used)
    fprintf('Fuel Used per Revolution = %6.5f (kg/Rev)
\n',Fuel_per_Rev)
    fprintf('Time per Cycle = %6.5f (days) \n',tf)
    fprintf('Time of Decay = %6.5f (days) \n',t1)
    fprintf('Time of Thrust = %6.5f (minutes) \n',t2)

end
```

PSOode

```
function [ xdot ] = PSOode( t,x )
%% State variables for PSO algorithm for decay
% 0 < t < t1

xdot = zeros(4,1);
```

```
%define s/c and orbit conditions
global muB
global Mass Cd A

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
end

%change density and Mass to canonical units
conv = density/Mass * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1);

end
```

PSOode2

```
function [ xdot ] = PSOode2( t,x )
%% State variables for PSO algorithm for thrust arc
% t1 < t < tf

global zeta0 zeta1 zeta2 zeta3 zeta4 zeta5 beta0 beta1 beta2 beta3
beta4 beta5
global muB
global c n0 Cd A v_Mass tlast integration

xdot = zeros(4,1);

%begin information about density
if (x(1)>=((140+6378.1)/6378.1)) && (x(1)<((200+6378.1)/6378.1))
    density=8*10^-7*exp(-.04*(x(1)-1)*6378.1);
elseif (x(1)>=((200+6378.1)/6378.1)) && (x(1)<((300+6378.1)/6378.1))
    density=3*10^-8*exp(-.024*(x(1)-1)*6378.1);
```

```
elseif (x(1)>=((300+6378.1)/6378.1)) && (x(1)<((400+6378.1)/6378.1))
    density=7*10^-9*exp(-.019*(x(1)-1)*6378.1);
else %condition for 400 to 500 km altitude
    density=3*10^-9*exp(-.017*(x(1)-1)*6378.1);
end

%change density and Mass to canonical units
n_t = n0*(beta0 + beta1*t + beta2*t^2 + beta3*t^3 + beta4*t^4 +
beta5*t^5);
v_Mass = v_Mass - n_t*(t-tlast);
integration = n_t*(t-tlast) + integration;
conv = density/(v_Mass) * (6378.1*1000)^3;

%state variables
xdot(1) = x(2);
xdot(2) = (-muB + x(1)*(x(1)*x(4))^2)/x(1)^2 -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(2) + ((c*n_t)/(c -
integration))*sin(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1) -
0.5*Cd*conv*sqrt(x(2)^2+(x(1)*x(4))^2)*A*x(1)*x(4)/x(1) + ((c*n_t)/(c -
integration))*cos(zeta0 + zeta1*t + zeta2*t^2 + zeta3*t^3 + zeta4*t^4 +
zeta5*t^5)/x(1);
tlast=t;

end
```

ACADEMIC VITA of Jack R. Chisholm

Jack R. Chisholm
86 Tenby Chase Drive
Voorhees, New Jersey, 08043
jack.r.chisholm@gmail.com

Education:

Bachelor of Science Degree in Aerospace Engineering
Penn State University, Spring 2012
Minor in Information Systems and Technology/Engineering Mechanics
Honors in Aerospace Engineering
Thesis Title: Particle Swarm Optimization Applied to Optimize Orbital Decay and Re-Boost Trajectories
Thesis Supervisor: Robert Melton

Royal Melbourne Institute of Technology – Spring 2011 Study Abroad

Related Experience:

Internship with Boeing in Ridley Park – Advanced Mobility Division
Worked with sizing Tilt-Rotors in the Phantom Works R and D
Supervisor: Steve Glusman
Summer 2011

Awards:

Leonhard Scholarship
Boeing Scholarship
Dean's List
Sigma Gamma Tau Aerospace Honors Society
Phi Eta Sigma National Honors Society

Presentations/Activities:

AIAA Regional Conference Presentation on Thesis
NASA's SISO Smackdown Competition Participant and Presenter
Orlando, March 2012
AIAA Penn State Chapter Member
Vice President 2011-2012 School Year
Engineering Orientation Network – Board of Directors