THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF AEROSPACE ENGINEERING

PARTICLE SWARM OPTIMIZATION APPLIED TO FINITE THRUST TRANSFERS BETWEEN TWO CIRCULAR NON-COPLANAR ORBITS

PETER FLANAGAN
SPRING 2016

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 Adviser

David B. Spencer
Professor of Aerospace Engineering
Faculty Reader

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

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

As a swarm intelligence scheme, the Particle Swarm Optimization (PSO) technique is a stochastic population-based method, representing an intuitive methodology for global optimization and has been successfully applied to several fields of research. Through mimicking the unpredictable motion of bird flocks in search of food, PSO uses the mechanism of information sharing that affects the overall behavior of a swarm to converge to the optimal values of the unknown parameters for the problem under consideration. For this research, PSO was used to optimize the finite thrust transfers of a spacecraft between two circular orbits that are not coplanar. The transfer trajectory consists of two thrusting arcs separated by a coasting arc. For better performance, the plane change was incorporated in the second thrusting maneuver. The dynamics of the system depend of the twelve coefficients from three cubic polynomials used to represent the in-plane and out-of-plane thrust pointing angles as well as the three time intervals corresponding to the three arcs of trajectory. Using MATLAB, the PSO algorithm will determine these fifteen parameters as the solution converges to the global optimal solution, minimizing the objective function, which corresponds to minimizing propellant consumption. The algorithm consists of eight functions, using ode45 to numerically integrate the state equations for each thrusting arc. Several tests were conducted on the PSO algorithm to analyze the convergence to the global minimum including varying the swarm parameters and the ratio of outer to inner radii values, β. Sometimes, the algorithm converged on a local minimum as the solution. Further research will attempt to correct the issue of local convergence, in hopes of consistently obtaining the global minimum.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

a = semi-major axis during the coasting arc

alpha = multipliers representing equality constraints for PSO algorithm

BestP = the P position that gave the particle its best value of J

BLp = the position lower bounds of the particle elements

BLv = the velocity lower bounds of the particle elements

BUp = the position upper bounds of the particle elements

BUv = the velocity upper bounds of the particle elements

c = effective thrust velocity of the propulsion system in DU/TU

c_C = cognitive weighting coefficient for UpdateV

c_I = inertial weighting coefficient for UpdateV

c_S = social weighting coefficient for UpdateV

d_k = penalty terms for equality constraints for the desired, or final, orbit

DU = canonical unit for distance (384,400 km)

E_1 = spacecraft's eccentric anomaly at t_1

E_2 = spacecraft's eccentric anomaly at t_2

e = eccentricity of the spacecraft's orbit during the coasting arc

f_1 = spacecraft's true anomaly at t_1

f_2 = spacecraft's true anomaly at t_2

GG = the best value of J for a given iteration

GGstar= array with the GG of each iteration

i_f = inclination of the spacecraft's final orbit in radians

J = objective function, the summation of the errors and the time intervals Δt_1 and Δt_2

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

k = this subscript denotes the integers 0,1,2,3

m_0 = initial mass of the spacecraft in MU

m_f = final mass of the spacecraft in MU

m = current mass of the spacecraft in MU

m_f/m_0 = the final mass to initial mass ratio

MU = canonical mass unit

N_elements = the number of unknown parameters each particle consists of

N_iterations = the number of iterations the PSO algorithm runs for

N_particles = the number of particles in the swarm the PSO algorithm analyzes

n_0 = the initial thrust-to-mass ratio of the spacecraft in DU/TU^2

ode45 = a built in numerical integrator in MATLAB

P = the dimensional array that holds the position for each particle and all of its elements

PSO = Particle Swarm Optimization

r = the radius of the spacecraft's orbit in DU

r_1 = the radius of the spacecraft's orbit at t_1 in DU

r_2 = the radius of the spacecraft's orbit at t_2 in DU

$\dot{r}$ = time derivative of the radius of the spacecraft's orbit in DU/TU

R1 = initial radius of the spacecraft's orbit in DU

R2 = final radius of the spacecraft's orbit in DU

t_0 = initial time the first thrust arc begins at in TU

t_1 = time where the first thrust arc ends in TU

t_2 = time where the second thrust arc begins at in TU

T = the spacecraft's thrust level

t_f = final time where the second thrust arc ends at in TU

tspan = the time span for the ODE functions

TU = canonical unit for time (375,190 s)

v_r = radial velocity component of the spacecraft in DU/TU

v_{r1} = radial velocity at t_1 in DU/TU

v_{r2} = radial velocity at t_2 in DU/TU

$\dot{v}_r$ = time derivative of the radial velocity component in DU/TU^2

v_θ = transverse velocity component of the spacecraft in DU/TU

$v_{\theta 1}$ = transverse velocity at t_1 in DU/TU

$v_{\theta 2}$ = transverse velocity at t_2 in DU/TU

$\dot{v}_\theta$ = time derivative of the transverse velocity component

z = distance of spacecraft normal to the initial plane

$\dot{z}$ = velocity of spacecraft normal to the initial plane

$\ddot{z}$ = out-of-plane equation of motion during the second thrust arc

α = out-of-plane thrust pointing angle during the second thrust arc

α_k = coefficients of the out-of-plane thrust pointing angle during second thrust arc

β = ratio of R2/R1

ΔE = eccentric anomaly variation

Δt_1 = time interval of the first finite thrusting maneuver

Δt_2 = time interval of the second finite thrusting maneuver

Δt_{CO} = coasting arc time interval in TU (t_2 - t_1)

Δv_1 = change in velocity during the first impulsive maneuver

Δv_2 = change in velocity during the second impulsive maneuver

δ = in-plane thrust pointing angle represented as a cubic polynomial

θ_k = coefficients of the second in-plane thrust pointing angle

ζ_k = coefficients of the first in-plane thrust pointing angle

θ = angular position in polar coordinates

$\dot{\theta}$ = time derivative of angular position in polar coordinates

$\ddot{\theta}$ = angular equation of motion of the spacecraft during the second thrust arc

μ_B = the gravitational parameter of attracting body in DU^3/TU^2

ξ = spacecraft angular displacement from the x-axis

ξ_1 =spacecraft angular displacement from the x-axis at t_1

$\dot{\xi}$ = time derivative of the angular displacement from the x-axis

ρ = radius in polar coordinates

$\dot{\rho}$ = time derivate of radius in polar coordinates

$\ddot{\rho}$ = radial equation of motion of the spacecraft during the second thrust arc

Chapter 1

Introduction

In orbital mechanics, it is crucial to perform an optimal orbital maneuver, whether it be for minimizing propellant consumption or time of the maneuver. The Hohmann transfer is a popular solution for calculating the minimum fuel needed for impulsive maneuvers between two coplanar circular orbits. An impulsive maneuver requires a truly instantaneous change in velocity, which is impossible. A finite, or non-impulsive, thrust is closer to resembling the physical nature/characteristics spacecraft actually employ, requiring numerical integration techniques to model. Low-thrust maneuvers that mimic Hohmann transfers between coplanar circular orbits have been calculated using Particle Swarm Optimization[1] . However, most LEO-GEO transfers require a plane-change, incorporating the full three-dimensional geometry of space.

In this thesis, the spacecraft will initially be traveling in a circular orbit around the Earth. Next, the spacecraft will begin its transfer trajectory, which is composed of two thrusting arcs, separated by a coasting arc. During the first thrusting arc and the coasting arc, the spacecraft remains in the initial plane. The plane change occurs entirely in the second thrusting arc to the inclination desired. Particle Swarm Optimization will search for the optimal solution with the minimal amount of propellant used, while ensuring the final orbital parameters are achieved.

Particle Swarm Optimization has been observed to converge at a local minimum in previous studies[1]. The results will document how well Particle Swarm Optimization is able to find the global minimum from testing various swarm parameters and radius ratios. Another area of interest in this thesis is comparing the finite thrusting maneuvers to impulsive ones for the same transfer, to demonstrate the limited performance attainable. The performance attainable by finite thrust is expected to approach the impulsive thrust approximation for higher thrust levels.

Particle Swarm Optimization

The Particle Swarm Optimization technique is a population-based stochastic method first introduced in 1995 to determine the optimal values of unknown parameters for a given problem. Successfully applied in several fields of research, Particle Swarm Optimization uses the mechanism of information sharing that affects the overall swarm. For the first iteration, the swarm population consists of randomly generated particles[1]. For each iteration, every particle is associated with its respective position and velocity vectors. The position vector is composed of the unknown parameters that are being solved for, and the velocity vector is the update to the position vector. Each particle is a potential solution, corresponding to a specific value of the objective function to be minimized. During each iteration, the best particle in the swarm, with the lowest objective function value, is selected. Once all of the iterations are complete, the best particle is found.

The particle swarm algorithm is defined by its simplicity and intuitiveness, being capable of finding the global optimal solution while requiring only the definition of the search space for the unknown parameters. The effectiveness of the algorithm increases with the number of particles in the swarm and/or the number of iterations[1]. Another factor in the success of PSO is the use of accelerator coefficients for updating the velocity vector, which increase in magnitude over the range of the iterations.

Using MATLAB, Particle Swarm Optimization will optimize the finite thrust transfers between two circular non-coplanar orbits. Each particle consists of fifteen elements, or the unknown parameters to be found. The first twelve elements are the coefficients of the three cubic polynomials for the in-plane thrust angles and the out-of-plane thrust angle. The final three elements are the thrusting time intervals and the variation in eccentric anomaly, which was used to derive the coasting time interval through Kepler's Law. The algorithm will minimize the objective function, which corresponds to the minimization of propellant and therefore the maximization of the final-to-initial mass ratio.

Chapter 2

Statement of the Problem

The purpose of this problem is to transfer a spacecraft initially in a circular orbit at R_1 to an outer orbit at R_2 of a different inclination angle. The initial conditions at t_0 are given by:

$$v_r(t_0) = 0 \tag{2.1}$$

$$v_\theta(t_0) = \sqrt{\left(\frac{\mu_B}{R_1}\right)} \tag{2.2}$$

$$r(t_0) = R_1 \tag{2.3}$$

$$\xi(t_0) = 0 \tag{2.4}$$

$$i(t_0) = 0 \tag{2.5}$$

where μ_B is the gravitational parameter of the attracting body, v_r, v_υ, r, and φ denote, respectively, the radial and the horizontal component of velocity, the radius, and the spacecraft angular displacement from the x-axis. The initial inclination of the orbit, i, is arbitrarily chosen to be zero. For practical purposes, Earth was chosen as the celestial body.

The trajectory of the spacecraft is broken down into three time intervals, two thrusting arcs separated by the coasting arc. The first thrusting arc begins at t_0, which is arbitrarily defined as zero, and ends at t_1. Then the thrust level is zero and the coasting arc begins, ending at t_2. At t_2, the second thrusting maneuver, which incorporates the plane change, begins until the final time t_f. Two additional assumptions are addressed. First, maximum thrust is assumed to be employed during the two thrusting maneuvers. Second, the in-plane and out-of-plane thrust

pointing angles are represented by cubic polynomial functions of time. With these assumptions, the thrust-to-mass ratio, T/m, is defined for the three time intervals by

Time Intervals:

$$0 \le t \le t_1 \qquad \frac{T}{m} = \frac{cn_0}{c - n_0 t} \tag{2.6}$$

$$t_1 \le t \le t_2 \qquad \frac{T}{m} = 0 \tag{2.7}$$

$$t_2 \le t \le t_f \qquad \frac{T}{m} = \frac{cn_0}{c - n_0(t_1 + t - t_2)} \tag{2.8}$$

where T is the thrust level, m is the current mass of the spacecraft and c, n_0, denote, respectively, the effective exhaust velocity of the propulsion system, and the thrust to mass ratio at t_0.

The spacecraft's motion for the first thrusting arc is defined by the following state equations:

$$\dot{v}_r = -\frac{\mu_B - r v_\theta^2}{r^2} + \frac{T}{m}\sin\delta \tag{2.9}$$

$$\dot{v}_\theta = -\frac{v_r v_\theta}{r} + \frac{T}{m}\cos\delta \tag{2.10}$$

$$\dot{r} = v_r \tag{2.11}$$

$$\dot{\xi} = \frac{v_\theta}{r} \tag{2.12}$$

where δ is the in-plane thrust pointing angle for $0 \le t \le t_1$.

$$\delta = \zeta_0 + \zeta_1 t + \zeta_2 t^2 + \zeta_3 t^3 \tag{2.13}$$

The unknown coefficients, $\{\zeta_0, \zeta_1, \zeta_2, \zeta_3\}$, are determined by the PSO algorithm, and initially are randomly generated within the search space.

The coasting arc trajectory is defined by the semi-major axis *a* and the eccentricity *e*.

$$a = \frac{\mu_B r_1}{2\mu_B - r_1(v_{r1}^2 + v_{\theta 1}^2)} \tag{2.14}$$

$$e = \sqrt{1 - \frac{r_1^2 v_{\theta 1}^2}{\mu_B a}} \tag{2.15}$$

where v_{r1}, $v_{\upsilon 1}$, r_1, and φ_1 are the values at t_1. With the trajectory being elliptical, the semi-major axis is greater than zero, the true anomaly at t_1, f_1, can be solved from the following two equations.

$$\sin f_1 = \frac{v_{r1}}{e}\sqrt{\frac{a(1-e^2)}{\mu_B}} \tag{2.16}$$

$$\cos f_1 = \frac{v_{\theta 1}}{e}\sqrt{\frac{a(1-e^2)}{\mu_B}} - \frac{1}{e} \tag{2.17}$$

The corresponding eccentric anomaly at t_1, E_1, is found by

$$\sin E_1 = \frac{sinf_1\sqrt{1-e^2}}{1+ecosf_1} \tag{2.18}$$

$$\cos E_1 = \frac{cosf_1 - e}{1-ecosf_1} \tag{2.19}$$

The eccentric anomaly at t_2 is $E_2 = E_1 + \Delta E$, where ΔE is one of the unknown parameters determined by the PSO algorithm. The corresponding true anomaly, f_2, is defined by

$$\sin f_2 = \frac{sinE_2\sqrt{1-e^2}}{1-ecosE_2} \tag{2.20}$$

$$\cos f_2 = \frac{cosE_2 - e}{1-ecosE_2} \tag{2.21}$$

The coasting time interval $\Delta t_{CO} \triangleq t_2 - t_1$ is derived from Kepler's Law

$$\Delta t_{CO} = \sqrt{\frac{a^3}{\mu_B}}[E_2 - E_1 - e(sinE_2 - sinE_1)] \tag{2.22}$$

For the second thrusting arc, the initial conditions are defined at time t_2

$$v_{r2} = \sqrt{\frac{\mu_B}{a(1-e^2)}}esinf_2 \tag{2.23}$$

$$v_{\theta 2} = \sqrt{\frac{\mu_B}{a(1-e^2)}}(1+ecosf_2) \tag{2.24}$$

$$r_2 = \frac{a(1-e^2)}{1+ecosf_2} \tag{2.25}$$

$$\xi_2 = \xi_1 + (f_2 - f_1) \tag{2.26}$$

The state equations for the second thrusting arc change, requiring two new states to define the out-of-plane motion $z, \dot{z}$. In polar coordinates, the equations of motion become

$$\ddot{\rho} = \rho\dot{\theta} - \frac{\mu_B \rho}{(\rho^2+z^2)^{\frac{3}{2}}} + \frac{T}{m}sin\delta\ cos\alpha \tag{2.27}$$

$$\ddot{\theta} = -\frac{2\dot{\rho}\dot{\theta}}{\rho} + \frac{T}{\rho m}cos\delta\ cos\alpha \tag{2.28}$$

$$\ddot{z} = -\frac{\mu_B z}{(\rho^2+z^2)^{\frac{3}{2}}} + \frac{T}{m}sin\delta \tag{2.29}$$

where δ is the in-plane thrust pointing angle for and α is the out-of-plane thrust pointing angle $t_2 \leq t \leq t_f$.

$$\delta = \theta_0 + \theta_1(t-t_2) + \theta_2(t-t_2)^2 + \theta_3(t-t_2)^3 \tag{2.30}$$

$$\alpha = \alpha_0 + \alpha_1(t-t_2) + \alpha_2(t-t_2)^2 + \alpha_3(t-t_2)^3 \tag{2.31}$$

Again, the unknown coefficients, $\{\theta_0, \theta_1, \theta_2, \theta_3, \alpha_0, \alpha_1, \alpha_2, \alpha_3\}$, are determined by Particle Swarm Optimization algorithm, and initially, during the first iteration are randomly generated with the search space.

The final conditions of the spacecraft are

$$v_r(t_f) = 0 \tag{2.32}$$

$$v_\theta(t_f) = \sqrt{\left(\frac{\mu_B}{R_2}\right)} \tag{2.33}$$

$$r(t_f) = R_2 \tag{2.34}$$

$$i(t_f) = i_f \tag{2.35}$$

where i_f is calculated by:

$$i_f = \cos^{-1}\frac{h_z}{|h|} \tag{2.36}$$

$$h = r\ x\ v \tag{2.37}$$

$$r = \rho\hat{\imath}_\rho + z\hat{\imath}_z \tag{2.38}$$

$$v = \dot{\rho}\hat{\imath}_\rho + \rho\dot{\theta}\hat{\imath}_\theta + \dot{z}\hat{\imath}_z \tag{2.39}$$

to ensure that final inclination condition is met.

The spacecraft system is dependent on twelve coefficients to represent the thrust pointing angle $\{\zeta_0, \zeta_1, \zeta_2, \zeta_3, \theta_0, \theta_1, \theta_2, \theta_3, \alpha_0, \alpha_1, \alpha_2, \alpha_3\}$, and the three time intervals Δt_1 ($\triangleq t_1$), Δt_{CO}, Δt_2 ($\triangleq t_f - t_2$). These unknown parameters are to be determined by Particle Swarm Optimization while finding the minimization of propellant, which corresponds to the maximization of the final-to-initial mass ratio m_f/m_0.

$$\frac{m_f}{m_0} = 1 - \frac{n_0}{c}(\Delta t_1 + \Delta t_2) \tag{2.40}$$

The impulsive Hohmann transfer outperforms the finite thrust transfer between two circular orbits for $\beta = R_2/R_1 < 11.939$. The Hohmann transfer has the final-to-initial mass ratio of

$$\left(\frac{m_f}{m_0}\right)_H = \exp\left[-\frac{\Delta v_1 + \Delta v_2}{c}\right] \tag{2.41}$$

where $\Delta v_1$and Δv_2 are calculated using the following equations:

$$\Delta v_1 = \left|v_p - v_{c1}\right| \tag{2.42}$$

$$\Delta v_2 = \sqrt{v_2^2 + v_{c2}^2 - 2v_2 v_{c2}\cos\Delta i} \tag{2.43}$$

$$v_2 = \sqrt{v_a^2 + v_{\Delta inc}^2} \tag{2.44}$$

$$v_{\Delta inc} = 2v_{c2}\sin\frac{\Delta i}{2} \tag{2.45}$$

with v_p, v_a, v_{c1}, v_{c1}defined as they are in a Hohmann transfer between two circular coplanar orbits[2].

Particle Swarm Optimization Problem Specific

Each particle in the swarm consists of the fifteen unknown parameters used $\{\zeta_0, \zeta_1, \zeta_2, \zeta_3, \theta_0, \theta_1, \theta_2, \theta_3, \alpha_0, \alpha_1, \alpha_2, \alpha_3, \Delta t_1, \Delta E, \Delta t_2\}$. The algorithm uses canonical units, where DU is the distance unit and TU is the time unit such that $\mu_B = 1\ DU^3/TU^2$. All the parameters are solved for within the defined search space, or range.

$$0\ TU \leq t_1 \leq 3\ TU \tag{2.46}$$

$$0 \leq \Delta E \leq 2\pi \tag{2.47}$$

$$0\ TU \leq t_2\ \leq 3\ TU \tag{2.48}$$

$$-1 \leq \zeta_k\ \leq 1 \tag{2.49}$$

$$-1 \leq \theta_k\ \leq 1 \tag{2.50}$$

$$-1 \leq \alpha_k\ \leq 1 \tag{2.51}$$

with k=0,1,2,3. The initial thrust-to-mass ratio n_0 is 0.16 DU/TU^2, and the effective exhaust velocity c is 0.5 DU/TU.

Four penalty terms, or equality constraints, are introduced to constrain the problem to an optimal solution that meets the final conditions, Eq.s 2.32-2.35. For the algorithm, an error less than 10^{-3} is considered acceptable, otherwise the error is multiplied by 100 in the objective function to be minimized.

Chapter 3

Method

The Particle Swarm Optimization Algorithm consists of eight MATLAB functions, aimed to minimize the objective function, find the corresponding optimal particle elements, and to display the results. Several common runtime errors may occur. For example, on occasion the PSO algorithm may converge to a local minimum instead of the global minimum. All optimization methods have this limitation of converging to a local minimum[1]. There is no guarantee of converging to the global minimum. Every time the code runs, it could converge to a different local minimum. For example, many times the code will converge to a result that satisfied only one of the equality constraints for the desired orbit. Therefore, for accurate results, the code should be run until all or most of the constraints are met. Another error that occurs is getting a negative semi-major axis. To circumvent this, for a particle with negative *a*, the objective function is set to infinity so the result does not converge to a value with a negative coast time. Figure 1 portrays the PSO algorithm in a flow chart.

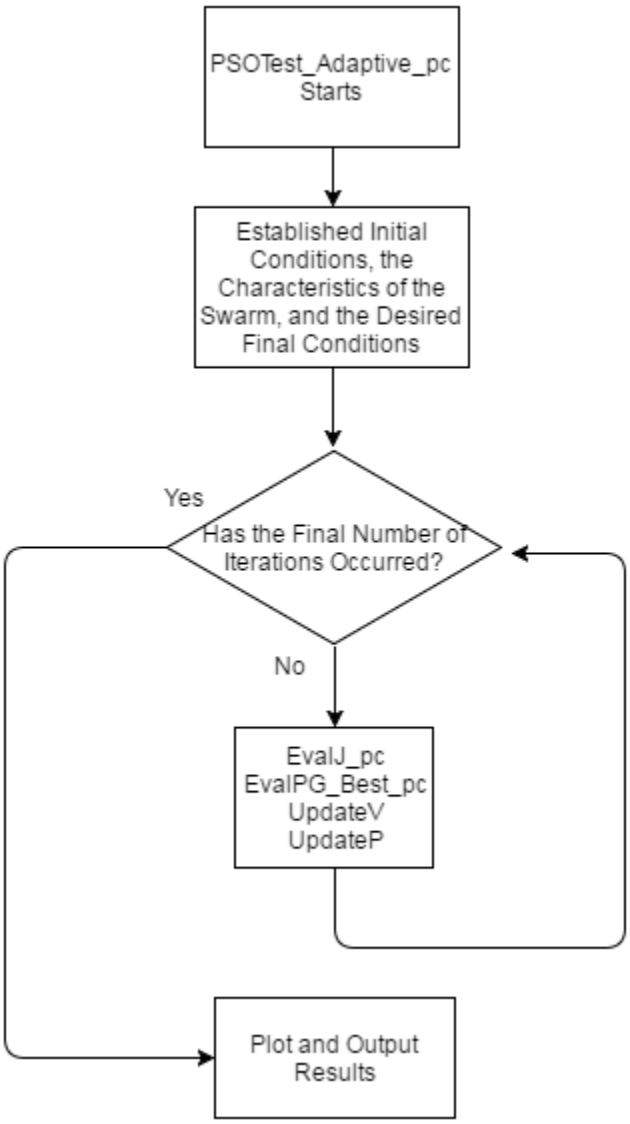

Figure 1. Flow Chart of PSO Algorithm

PSOTest_Adaptive_pc

PSOTest_Adaptive is the main function of the MATLAB code. The main function first establishes the characteristics of the swarm population (i.e. the number of unknown elements in a particle, the number of particles, and the number of iterations), the initial conditions of the spacecraft, the position vector's upper and lower bounds, the velocity vector's upper and lower

bounds, and finally the random swarm population within the allocated boundaries. For this thesis, the number of unknown elements to be solved is 15, and unless otherwise noted uses a swarm of 30 particles and 500 iterations. The upper bounds of the position and velocity vectors are denoted, BUp and BUv respectively. Likewise, the lower bounds of the position and velocity vectors are denoted, BLp and BLv respectively. These bounds correspond to within the defined search space for the unknown parameters, Eqs. 2.46- 2.51.

Next, the function creates the sizes of the PBest, J, JBest, and V matrices, filling them with zeros. The values in JBest array and GG are set to infinity, so that any finite value to the objective function will take its place as the best solution in the first iteration.

The main function then uses seven other function to determine the optimal solution. First, Impulse.m is called to find the optimal performance attainable through an idealized Hohmann transfer. This is used to compare the finite thrust to the idealized impulsive thrust case. Next, a loop is used to run through six functions for a total of N_iterations. For each iteration, the following steps occur. First, every particle in the swarm is evaluated, where the one with the best performance (lowest J) is found. The particle is then compared to the best position found from the previous iterations, and sets that particle to the new best particle if it's GG value is lower. The loop then prepares for the next iteration by updating the velocity and position vectors. To observe the results in real time, the code displays Δt_1, Δt_{CO}, Δt_2, $\Delta\varphi$, and the Mass Ratio, and finds the error between the current GGBest value and the idealized. This process is repeated for every iteration. Once this loop is completed, several results are plotted

EvalJ_pc

EvalJ_pc.m is the first function called for every iteration, which calculates the objective function for all of the particles using a loop for a total of N_particles. The function is broken down into three time intervals the first thrusting arc, the coasting arc, and the second thrusting arc. The first four unknown elements, the coefficients of the first in-plane thrusting angle, are read into the function, as well as the final three, Δt_1, ΔE, Δt_2. Next, EvalJ_pc.m calls ode45 on another function, SCEoM1, to numerically integrate the spacecraft's differential equations of motion during the first thrusting arc, for a time span from t_0 to t_1, (Δt_1). The numerical integrator, ode45 uses absolute and relative tolerances of 1e-6. Once completed, EvalJ_pc.m takes the spacecraft's final state of motion in the thrusting arc and begins the coasting arc, using those values to find the semi-major axis, eccentricity, true anomaly variation, and eccentric anomaly variation. EvalJ_pc.m takes the final conditions of the coasting arc and uses them as initial conditions for the second thrusting arc. Again, ode45 is used on another function, SCEoM_pc, to solve for the spacecraft's motion during the seconding thrusting arc (which incorporates the change in inclination), for a time span from t_2 to t_f, Δt_2. From the final state, t_f, the final inclination is found. Four inequality constraints are used to ensure the final conditions are met within an error of 10^-3, an acceptable tolerance. These constraints introduce four penalty terms to be summed with the objective function to attempt to act as an avoidance to solutions that do not meet with the desired final conditions.

$$d_1 = v_r(t_f) \tag{3.1}$$

$$d_2 = v_\theta(t_f) - \sqrt{\frac{\mu_B}{R_2}} \tag{3.2}$$

$$d_3 = r(t_f) - R_2 \tag{3.3}$$

$$d_4 = inc_f - inc_{28.5^\circ} \tag{3.4}$$

$$J = \Delta t_1 + \Delta t_2 + \sum \alpha_k |d_k| \tag{3.5}$$

EvalPGBest_pc

EvalPGBest_pc.m determines the best position within the iteration and the best position visited by a particle up to the current iteration. The function stores the minimum objective function value found within the current iteration into JBest, as well as the corresponding the best particle, PBest. The GG value, the best performance within all iterations, is stored, as well as the corresponding best particle. For the best particle, the mass ratio, Δt_{CO}, and $\Delta\varphi$ values are calculated and stored.

UpdateV

After EvalPGBest_pc.m, UpdateV.m is called to update the velocity vector V for the particles.

$$v_i = c_I v_i + c_c v_i = c_i v_i + c_C(PBest_i - P_i) + c_S(GBest - P_i) \tag{3.6}$$

where c_I, c_C, and c_S are accelerator coefficients which increase in magnitude over the entire range of N_iterations.

$$c_I = \frac{1 + rand}{2} \tag{3.7}$$

$$c_C = 0.01 + \frac{1.49445 * rand * j}{N\_iterations} \quad (3.8)$$

$$c_S = 0.01 + \frac{1.49445 * rand * j}{N\_iterations} \quad (3.9)$$

where rand is a random number between 0 and 1, and j is the current iteration. The inertia coefficient, c_I, determines how much the velocity remains unchanged. The cognitive coefficient, c_C, determines how much the velocity should adjust to make each particle P move closer to PBest, the best in the iteration. The social coefficient, c_S, determines how much the velocity should adjust to make each particle P move to GBest, the globally best particle. Next, UpdateV.m checks to makes sure every element is within the velocity bounds. If outside either bound, the element is set to the value of the boundary it violated.

UpdateP

The UpdateP.m function updates the position, particle elements, of each particle by adding the new velocity vector to the previous position vector. UpdateP.m also checks to make sure that every element is within the position bounds. If outside, the element's position is set to the value of the particle boundary it violated.

Impulsive

The Impulsive.m function's purpose is to compare the optimal performance attainable through an idealized Hohmann transfer. This performance demonstrates the limiting performance attainable by any finite thrust when compared to the idealized impulsive thrust case. Impulsive.m simply calculates the mass ratio for an impulsive Hohmann transfer for non-coplanar circular

obits. Note the impulsive mass ratio will always be greater but the finite mass ratio should be close to this approximation.

SCEoM1

SCEoM1.m calculates the spacecraft's trajectory using the state equations for the first thrusting arc. No out-of-plane thrust occurs here, as it would be less efficient to perform the out-of-plane Δv when you are closer to the attracting body. The spacecraft starts with the initial conditions established in the main function, and determines the motion of the spacecraft for the first time span, Δt_1. At t_1, the propulsive system shuts down, and SCEoM.1 ends. This function runs for every particle in every iteration.

SCEoM_pc

SCEoM_pc.m calculates the spacecraft's trajectory using state equations that correspond to the equations of motion described for the second thrusting maneuver. The spacecraft's motion begins with the final conditions from the coasting arc, and the motion is calculated for the second time span, Δt_2. This function runs for every particle in every iteration.

Chapter 4

Results

The results from the PSO algorithm are discussed in this section. First, the ratio of the radii, β, is tested to observe its effect on the objective function's convergence. Several other tests were conducted using different swarm parameters to observe their effect on converging to the global optimal solution as well. The swarm parameters being tested are the number of particles in the swarm, and the number of iterations.

For all of the tests, the globally optimal solution found was when the final value, GGBest, met all four of the desired orbit constraints. Again, it is worth mentioning that the algorithm could converge to a local minimum instead. The range of this error was significant, making it necessary for multiple runs to ensure the smallest error. Running the algorithm found that the amount of runs needed varied greatly, and was dependent on the swarm parameters and β. Finally, the final-to-initial mass ratios for finite thrust are compared to the ideal impulsive transfer, which constitutes the limiting performance attainable by any finite thrust transfer.

Varying the radii ratio, β

The optimal solution occurs when the best value of the objective function up to the current iteration, GGstar approximately equates to $\Delta t_1 + \Delta t_2$. This indicates that all of the final orbital conditions were met within the tolerance of 10^-3, therefore no equality constraints are violated. As mentioned in the common runtime errors, achieving the global minimum takes multiple trial runs. For instance, when the radii ratio β= 2, a very good result was achieved after

running the PSO algorithm 9 times as shown in Fig. 1, where each run for β=2 took approximately 5 minutes. For the next two figures, N_iterations =300, N_particles =30.

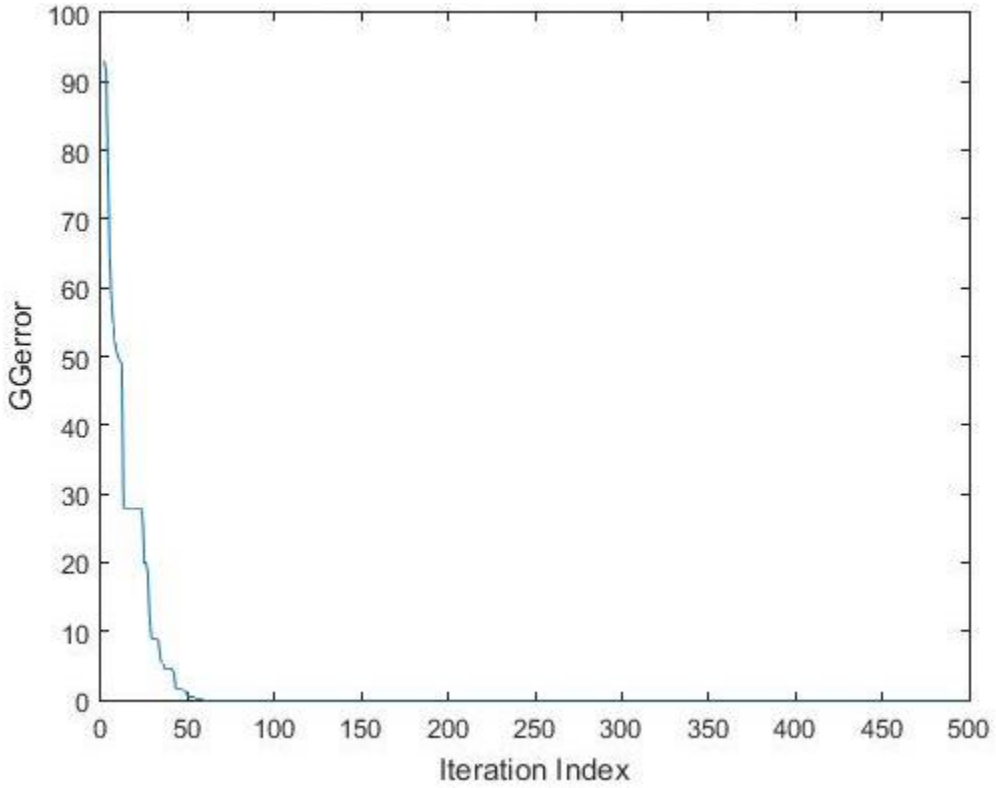

Figure 2. GGError vs Number of Iterations (β=2)

As β increased, the amount of runs it takes to converge to the global minimum and the run time increase. Figure 2 demonstrates the optimal solution being found, as the error approaches to and becomes zero at N_iterations= 59. This result was achieved within 15 trial runs. On the other hand, the results from Figure 3 which also indicate that the global minimum was reached within 35 trial runs.

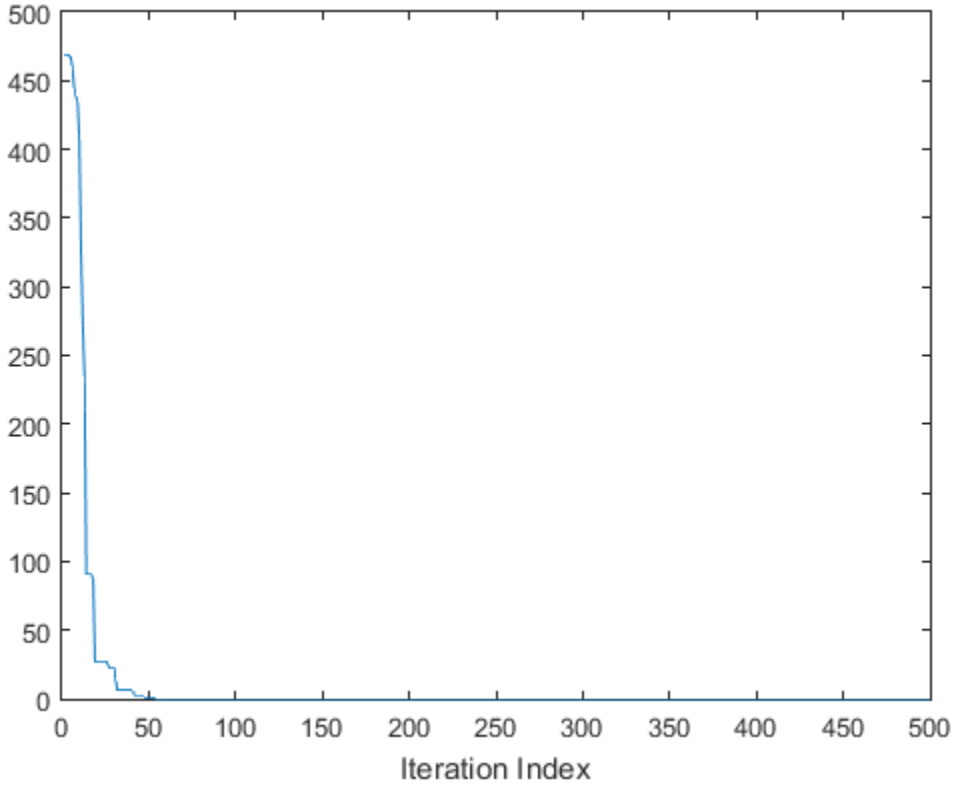

Figure 3. GGError vs Number of Iterations (β=6)

The algorithm was used for all even β values within the bitangent Hohmann transfer region ($1< R_2/R_1 < 11.939$). In several cases, the global minimum could not be found within the amount of trial runs tested. Instead, the results for the local minimum that were the closest to the global solution were used. Table 1 shows the optimal results obtained for several different radius ratios.

Table 1. Best Numerical Results for different β for 30 Runs

β	Δt_1, TU	Δt_{CO}, TU	Δt_2, TU	$\Delta\xi$, deg	GGBest	Error	# of α_k =100
2	0.80155	3.06699	1.95030	111.72359	2.75185	0	0
4	1.40976	7.73471	1.54930	109.00075	3.07958	0.12052	1
6	1.55801	38.23609	0.94149	151.42888	2.49950	0	0
8	1.56717	30.49002	0.56933	134.97580	3.98331	1.84681	2
10	1.68339	37.30127	0.86023	137.42567	3.48882	0.94520	2

where Error is defined by Eq. 4.1 and # of α_k = 100 refers to how many equality constraints have been violated.

$$Error = GGBest - \Delta t_1 + \Delta t_2 \quad (4.1)$$

When the objective function converges to a result with no violation of the equality constraints, then the global optimal solution has been found and no further runs are required. However, the global solution was only achieved for radii ratios of 2 and 6 out of all the runs completed. In the cases where β was equal to 4,8, and 10, the global solution was not found, as the solution converged to a local minimum where 1 or 2 of the 4 final conditions were violated. Figure 4 demonstrates this occurrence for $\beta = 4$.

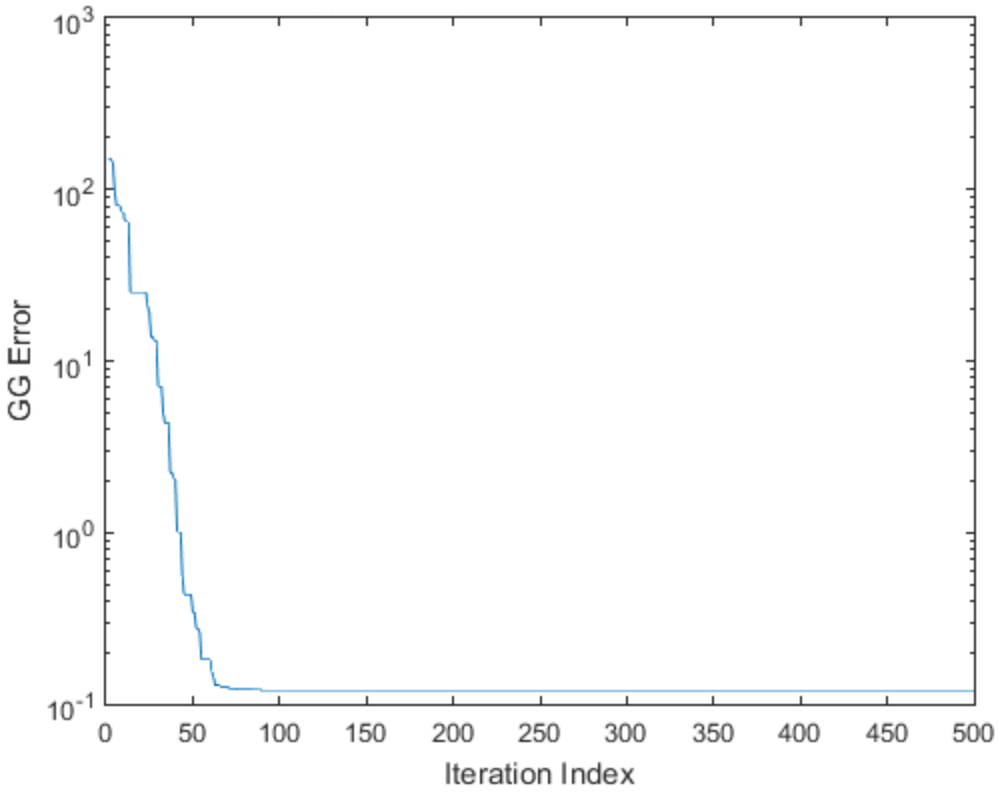

Figure 4. Log(GGError) vs Number of Iterations (β= 4)

Varying the Number of Particles, N_particles

Varying the particles of the swarm affected both the time it takes for the code to run and the chances of the code converging to the global minimum. Increasing the number of particles allows the global minimum to be usually found in fewer run times. Usually since there is a random element in the optimization process. Therefore using a greater amount of particles was mostly found to increase the run time significantly for a tradeoff of better results. Exceptions to this, as demonstrated in Fig. 4, can occur since the global minimum is not always achieved.

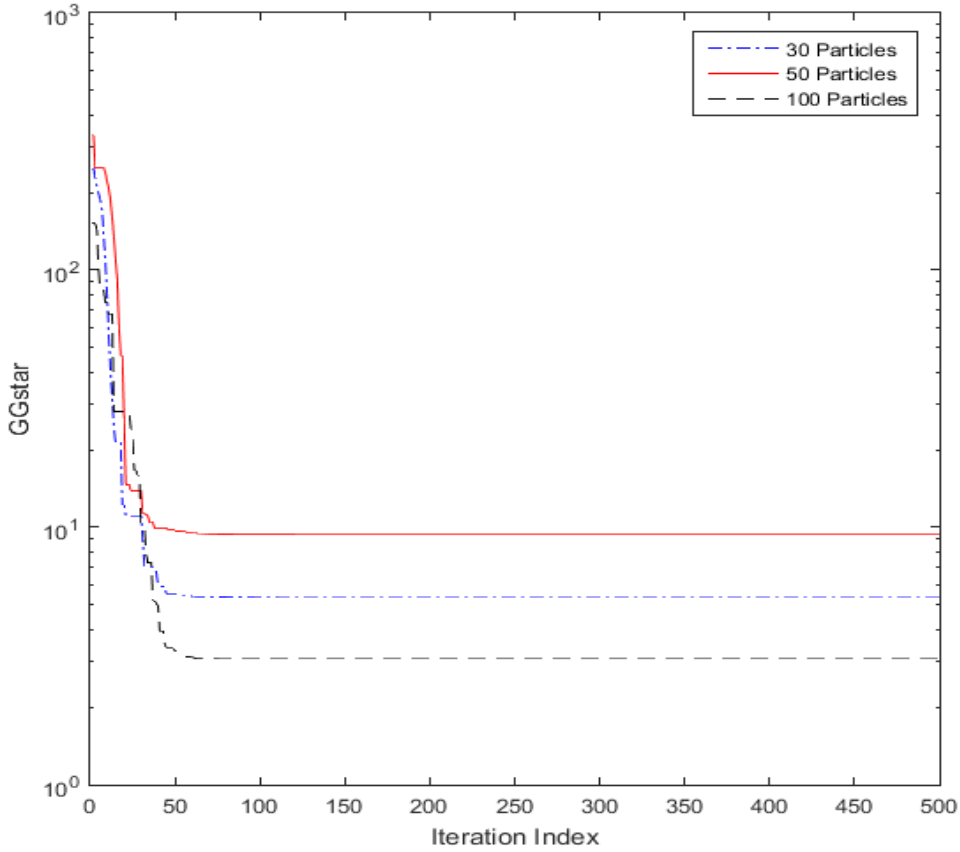

Figure 5. Particles Comparison for 500 Iterations and β = 4

As mentioned previously, increasing the number of particles also increased the time for the algorithm to complete. For example particles that were varied 30, 50, 100 for β=4 in Figure

5, respectively corresponded to run times of 133.7 s, 245.6 s, 857.6 s. This trend was found to be consistent for amongst all results the various radii ratios.

Varying the Number of Iterations, N_iterations

Varying the number of iterations also affected the run time and performance. Using a greater amount of iterations was found to increase the run time significant with the tradeoff of better results. For instance, the number of iterations was varied 250, 400, 500 for 30 particles at $\beta=4$. This trend was found to be common amongst all radii ratios. An example is shown in Figure 6, demonstrating the more iterations the better performance observed.

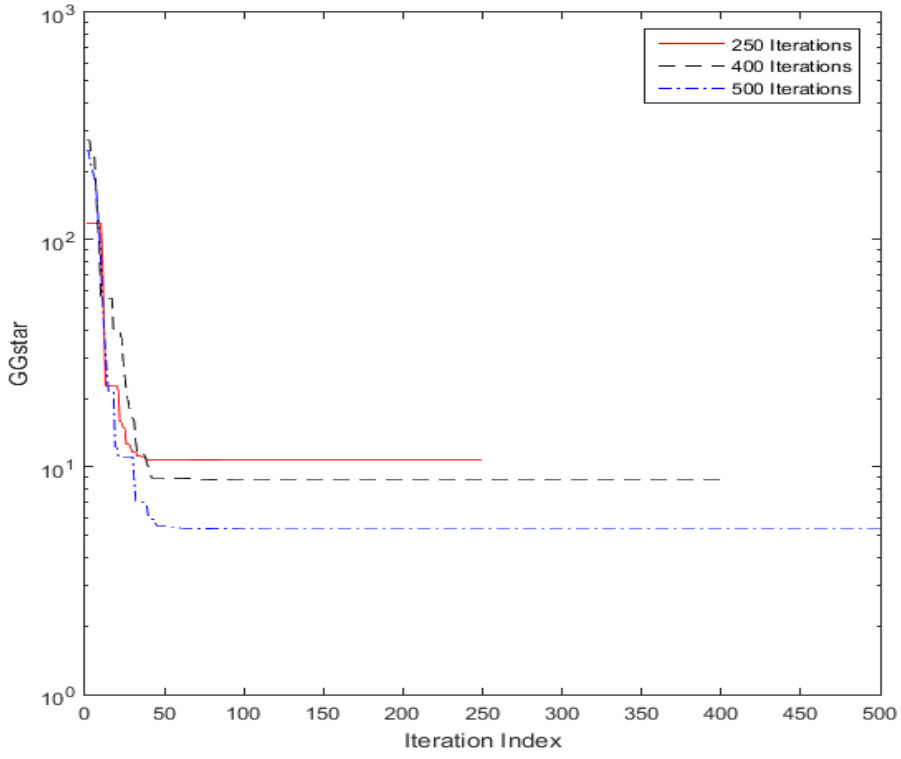

Figure 6. Iterations Comparison for 30 Particles and β = 4

Mass Ratio: Finite vs Impulsive

The performance attainable from the optimal finite thrust transfer is expected to approach the impulsive thrust approximation as the thrust level increases. The impulsive transfer constitutes the limiting performance attainable by any finite transfer[1]. Due to this fact, the results between the two transfer types are compared in Table 2.

Table 2. Mass Ratio: Finite vs Impulsive Thrust for GGBest

β	Δt_1, TU	Δt_{CO}, TU	Δt_2, TU	$\Delta\xi$, deg	GGBest	$\frac{m_f}{m_0}$	$\frac{m_f}{m_{0_I}}$
2	0.80155	3.06699	1.95030	111.72359	2.75185	0.11941	0.37068
4	1.40976	7.73471	1.54930	109.00075	3.07958	0.05310	0.36305
6	1.55801	38.23609	0.94149	151.42888	2.49950	0.20016	0.35789
8	1.56717	30.49002	0.56933	134.97580	3.98331	0.31632	0.35638
10	1.68339	37.30127	0.86023	137.42567	3.48882	0.18604	0.35660

Unlike Pontani and Conway's results, the final-to-initial mass ratios of the global and local minima do not approach the impulsive estimates. This result is interesting, because the minimization of the objective function should correlate to the maximization of the final-to-initial mass ratio. It is noteworthy that higher values of the final-to-initial mass ratio for finite thrust have been found while the algorithm converges to a local minimum.

Chapter 5

Conclusions and Future Work

Many orbital maneuvers require a plane change, such as LEO-GEO transfers. As opposed to an impulsive transfer, an optimal finite thrust solution is used to simulate a more realistic model the nature of spacecraft. Particle Swarm Optimization is useful in optimizing space trajectories. This research shows that Particle Swarm Optimization can be used to find the global minimum after enough trial runs. The Particle Swarm Optimization algorithm can converge to a local minimum where only part of the equality constraints are fulfilled. Finding the global minimum can be a grueling process, and requires some luck and time to run multiple trials. The theorized correlation between better performance achieved from both a larger number of particles in the swarm and a larger number of iterations was confirmed.

In the future, the code should be further analyzed to solve some of its issues. One problem is that early on in some trial runs occurs when the code finds a particle that is within the tolerance for all four equality constraints, but it fails all four in the next iteration. Potentially, a feature would be introduced to keep those particle elements unchanged whenever a particle is within all four tolerances. Any way to cut down on the amount of attempts and time needed to find the global minimum would be highly beneficial. Another suggestion would be to run the tests with more iterations and particles to achieve better results, see if converges to global minimum in less trial runs. Finally, it would be interesting to find out why there is such a significant difference in mass ratios results between finite and impulsive. Unlike the ones achieved in Potani and Conway[1], which predicts significantly closer mass ratios.

Appendix A

PSO Algorithm

PSOTest_adaptive_pc

```
%% PSO With Variable Accelerator Coefficients
%%
clc;
clear all;
close all;

%To Turn Off Warnings
% [a, MSGID] = lastwarn();
% warning('off', MSGID);

global P J JBest PBest GG N_particles N_elements V BLv BUv BLp BUp mu R1 R2
global N_iterations
global c n0 DU TU vrt0 vtt0 rt0 angt0 vrtf vttf rtf theta0 i_tf dt_coast
global MassRatio MassRatio_I BestP GBest d_ang j %L2 endlimit L1 x y z

N_particles = 100;
N_elements = 15;
N_iterations = 500;

mu=1;
R1=1;
R2=10;

DU=R1; %distance canonical unit
TU=sqrt(DU^3/mu); %time canonical unit
c=0.5*DU/TU; %effective exhaust velocity
n0=0.16*DU/TU^2; %thurst to mass ratio
i_f= 28.5*pi/180; %degrees converted to rads
%i_f=0;

%Initial and Final Conditions
vrt0=0;
vtt0=sqrt(mu/R1);
rt0=R1;
angt0=0; %angular displacement from x axis
theta0=0;
vrtf=0;
vttf=sqrt(mu/R2);
rtf=R2;
i_tf= i_f;
```

```
%%
%% set lower and upper bounds on unknowns (particle elements)
BLp = [-1, -1, -1, -1, -1, -1, -1, -1, -0.5*pi,-0.5*pi,-0.5*pi,-0.5*pi, 1e-5,
1e-5, 1e-5];
BUp = [1, 1, 1, 1, 1, 1, 1, 1, 0.5*pi, 0.5*pi, 0.5*pi, 0.5*pi, 3*TU, 2*pi-1e-
5, 3*TU];
%% create random initial population
for i=1:N_particles
    P(i,1)= BLp(1)+rand*(BUp(1)-BLp(1));     %zeta0 coefficients during first
thrusting maneuver
    P(i,2)= BLp(2)+rand*(BUp(2)-BLp(2));     %zeta1 these are the
coefficients of the thrusting angle polynomial
    P(i,3)= BLp(3)+rand*(BUp(3)-BLp(3));     %zeta2
    P(i,4)= BLp(4)+rand*(BUp(4)-BLp(4));     %zeta3
    P(i,5)= BLp(5)+rand*(BUp(5)-BLp(5));     %zeta0 coefficients during
second thrusting maneuver
    P(i,6)= BLp(6)+rand*(BUp(6)-BLp(6));     %zeta1
    P(i,7)= BLp(7)+rand*(BUp(7)-BLp(7));     %zeta2
    P(i,8)= BLp(8)+rand*(BUp(8)-BLp(8));     %zeta3
    P(i,9)= BLp(9)+rand*(BUp(9)-BLp(9));     %alpha0
    P(i,10)= BLp(10)+rand*(BUp(10)-BLp(10)); %alpha1
    P(i,11)= BLp(11)+rand*(BUp(11)-BLp(11)); %alpha2
    P(i,12)= BLp(12)+rand*(BUp(12)-BLp(12)); %alpha3
    P(i,13)= BLp(13)+rand*(BUp(13)-BLp(13)); %dt1
    P(i,14)= BLp(14)+rand*(BUp(14)-BLp(14)); %dE
    P(i,15)= BLp(15)+rand*(BUp(15)-BLp(15)); %dt2
end

% for i=1:N_particles
%     P(i,:)= BLp+rand*(BUp-BLp); %something is wrong here
% end

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

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

Impulsive;

tic;
for j = 1:N_iterations
    EvalJ_pc;
    EvalPGBest_pc;
    UpdateV(j);
    UpdateP;
    GGstar(j) = GG;
```

```
    fprintf('GG = %6.5f \n',GG)
    fprintf('dt1 = %6.5f, dtcoast = %6.5f, dt2 = %6.5f, dAng = %6.5f
\n',GBest(13),dt_coast,GBest(15), d_ang*180/pi)
    fprintf('Mass Ratio = %6.5f \n', MassRatio);
    GGerror(j)= abs(GG-(GBest(13)+GBest(15)));
    fprintf('Iteration : %1.0f \n', j)
    disp(' ')
end
save 'PSO_adapt' GGstar, N_iterations;
toc;
fprintf('Best Particle = %6.5f \n', BestP);

finalerror= abs(MassRatio - MassRatio_I);
fprintf('Mass Difference = %6.5f \n', finalerror);
FinalGGerror= abs(GG-(GBest(13)+GBest(15)));
fprintf('Final Error = %6.5f \n', FinalGGerror);

%% Plots
figure()
plot(1:N_iterations, GGstar)
title('GGstar vs Number of Iterations');
ylabel('GGstar');
xlabel('Iteration Index')
% hold on;

figure()
ispan= linspace(1,N_iterations,N_iterations);
semilogy(ispan,GGstar)
title('Log(GGstar) vs Number of Iterations');
ylabel('GGstar');
xlabel('Iteration Index')
% hold on;

figure()
plot(1:N_iterations,GGerror(1:N_iterations))
title('GGerror vs Number of Iterations');
ylabel('GGstar');
xlabel('Iteration Index')
% hold on;

figure()
ispan= linspace(1,N_iterations,N_iterations);
semilogy(ispan,GGerror(1:N_iterations))
title('Log(GGError) vs Number of Iterations');
ylabel('GG Error');
xlabel('Iteration Index')

% Plots the spacecraft's motion through the 3 arcs
% figure()
% x(1:L1,5)=0;
% plot3(x(1:L1,1),x(1:L1,3),x(1:L1,5),'-r') %solid line red
% hold on;
%
% y(1:endlimit,5)=0;
```

```
% plot3(y(1:endlimit,1),y(1:endlimit,3),y(1:endlimit,5),'--g') %dashed green
% hold on;
%
% plot3(z(1:L2,1),z(1:L2,3),z(1:L2,5), '-.b') %dash dot blue
% hold on;
%
% grid on;
% axis equal;
```

EvalJ_pc

```
function EvalJ_pc()
%% EvalJ evaluates J for each particle in current iteration
global P J N_particles mu R1 R2 a e %ft vrfminus vtfminus dv2 vt0minus
vt0plus vr0plus
global zeta0 zeta1 zeta2 zeta3 dt1 dE dt2 dtcoast t2 tf
global alpha0 alpha1 alpha2 alpha3 i_tf
global vrt0 vtt0 rt0 angt0 %vrtf vttf rtf
global angt2 theta_1 %x y z j f endlimit_f endlimit L1 L2

for i = 1:N_particles
    %Begin First Thrusting Arc
    zeta0=P(i,1);
    zeta1=P(i,2);
    zeta2=P(i,3);
    zeta3=P(i,4);
    dt1=P(i,13);
    dE=P(i,14);
    dt2=P(i,15);

    x=[rt0;vrt0;angt0;vtt0];
    tspan= [0,dt1];
    options= odeset('RelTol',1e-6,'AbsTol',1e-6);
    [t,x]=ode45('SCEoM1',tspan,x,options);
    %[t,x]=ode23s('SCEoM1',tspan,x,options);
    L1= length(x);


    %Final State of Motion at End of First Thrusting Arc
    r_t1= x(L1,1);
    v_rt1= x(L1,2);
    theta_1(i)= x(L1,3); %theta_1 is an array so EvalPGBest can use the best
value for result discussion
    while (theta_1(i)<0 || theta_1(i)>(2*pi))
        if theta_1(i)<0
            theta_1(i)=theta_1(i)+(2*pi);
        elseif theta_1(i)>(2*pi)
            theta_1(i)=theta_1(i)-(2*pi);
        end
```

```
    end
    v_tt1= x(L1,4);


    %% Coasting Arc
    a= (mu*r_t1)/((2*mu)-(r_t1*(v_rt1^2+v_tt1^2)));
    e= sqrt(1-((r_t1^2*v_tt1^2)/(mu*a)));

    yf1=(v_rt1/e)*sqrt(a*(1-e^2)/mu);           % sin f1
    xf1=((v_tt1/e)*sqrt(a*(1-e^2)/mu))-(1/e);   % cos f1
    f1=atan(yf1/xf1);
    if xf1<0
        f1=f1+pi;
    end
    while (f1<0 || f1>(2*pi))
        if f1<0
            f1=f1+(2*pi);
        elseif f1>(2*pi)
            f1=f1-(2*pi);
        end
    end

    yE1=(sin(f1)*sqrt(1-e^2))/(1+e*cos(f1));    % sin E1
    xE1=(cos(f1)+e)/(1+e*cos(f1));              % cos E1
    E1= atan(yE1/xE1);
    E1i= E1;
    if xE1<0
        E1=E1+pi;
    end

    E2=E1+dE;

    yf2=(sin(E2)*sqrt(1-e^2))/(1-e*cos(E2));    % sin f2
    xf2=(cos(E2)-e)/(1-e*cos(E2));              % cos f2
    f2= atan(yf2/xf2);
    if xf2<0
        f2=f2+pi;
    end
    while (f2<0 || f2>(2*pi))
        if f2<0
            f2=f2+(2*pi);
        elseif f2>(2*pi)
            f2=f2-(2*pi);
        end
    end

    rt2=(a*(1-e^2))/(1+e*cos(f2));
    vrt2=sqrt(mu/(a*(1-e^2)))*e*sin(f2);
    angt2(i)=(theta_1(i)+(f2-f1));%angt2 is an array so EvalPGBest can use
the best value for results
    vtt2=sqrt(mu/(a*(1-e^2)))*(1+e*cos(f2));

    %Final Conditions for Coasting Arc
    rho= rt2;
```

```
    rho_dot= vrt2;
    theta= angt2(i); %angt2 is an array so EvalPGBest can use the best value
for results
    theta_dot= vtt2/rt2;
    z=0;
    z_dot=0;

    %%
    dtcoast(i)= sqrt(a^3/mu)*(dE-e*(sin(E2)-sin(E1)));
%     if dtcoast(i)<0
%         fprintf('Error: Negative Coast Time for = %1.0f \n', i);
%         dtcoast(i)=abs(dtcoast(i));
%     end
    t2=dt1+dtcoast(i);
    tf=dt2+t2;

    %% Ignore. This was used to visualize the orbit during coastic/check
aboce values
%     xt = [x(L1,1); x(L1,2);x(L1,3);x(L1,4)];
%     tspanc = [0 dtcoast(i)];
%     options=odeset('RelTol', 1e-6, 'AbsTol', 1e-6);
%     [t,y] = ode45('SCEoM_co',tspanc,xt,options);
%     endlimit = length(y);
%
%     rhot= y(endlimit,1);
%     rho_dott= y(endlimit,2);
%     thetat= y(endlimit,3);
%     while (thetat<0 || thetat>(2*pi))
%         if thetat<0
%             thetat=thetat+(2*pi);
%         elseif thetat>(2*pi)
%             thetat=thetat-(2*pi);
%         end
%     end
%     vtt2t = y(endlimit,4);
%     theta_dott= vtt2t/rhot;

    %% Begin Second Thrusting Arc
    zeta0=P(i,5);
    zeta1=P(i,6);
    zeta2=P(i,7);
    zeta3=P(i,8);
    alpha0= P(i,9);
    alpha1= P(i,10);
    alpha2= P(i,11);
    alpha3= P(i,12);

    x2=[rho;rho_dot;theta;theta_dot;z;z_dot];
    tspan2= [t2,tf];
    options= odeset('RelTol',1e-6,'AbsTol',1e-6);
    [t,z]=ode45('SCEoM_pc',tspan2,x2,options);
    %[t,z]=ode23s('SCEoM_pc',tspan2,x2,options);
    L2= length(z);

    r= [z(L2,1); 0; z(L2,5)];
```

```
    v= [z(L2,2); z(L2,1)*z(L2,4); z(L2,6)];
    h=cross(r,v);
    hmag= (dot(h,h))^(1/2);
    K=[0;0;1]; %z component
    inclination=acos(dot(h,K)/hmag);

    while (inclination<-1e-3|| inclination>2*pi)
        if inclination<-1e-3
            inclination=inclination+2*pi;
        elseif inclination>2*pi
            inclination=inclination-2*pi;
        end
    end

    %% Compare Section: PENALTIES USE To Desired final orbit paramaters

%Penalty Terms
    r_tf= z(L2,1);
    v_rtf= z(L2,2);
    theta_f= z(L2,3);
    theta_dotf= z(L2,4);

    v_ttf = r_tf*theta_dotf;

    d(1)=v_rtf;
    d(2)=v_ttf-sqrt(mu/R2);
    d(3)=r_tf-R2;
    d(4)=inclination-i_tf;
    if abs(d(1))>10^-3
        alpha_1=100;
    else
        alpha_1=0;
        fprintf('Good alpha1 Value for = %1.0f \n', i);
    end
    if abs(d(2))>10^-3
        alpha_2=100;
    else
        alpha_2=0;
        fprintf('Good alpha2 Value for = %1.0f \n', i);
    end
    if abs(d(3))>10^-3
        alpha_3=100;
    else
        alpha_3=0;
        fprintf('Good alpha3 Value for = %1.0f \n', i);
    end
    if abs(d(4))>10^-3
        alpha_4=100;
    else
        alpha_4=0;
        fprintf('Good alpha4 Value for = %1.0f \n', i);
    end

    if a > -1e-3 %0
```

```
        J(i) = dt1 + dt2 + alpha_1*abs(d(1))+ alpha_2*abs(d(2))+
alpha_3*abs(d(3)) + alpha_4*abs(d(4)); %switched dt1 and dt2 instead of P's
    else
        J(i)= inf;
    end
end

end
```

EvalPGBest_pc

```
function EvalPGBest_pc()
%% EvalP&GBest determines the best position visited by particle i up through
%%   the current iteration)
global P J JBest PBest GG GBest N_particles BestP n0 c MassRatio dtcoast
dt_coast
global d_ang angt2 theta_1

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,:);
       BestP= i;
       MassRatio = 1-((n0/c)*(GBest(13)+GBest(15)));
       dt_coast = dtcoast(i);
       d_ang = angt2(i)-theta_1(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
global N_iterations
c_I = (1 + rand)/2;
c_C = 0.01 + 1.49445*rand*j/N_iterations;
```

```
c_S = 0.01 + 1.49445*rand*j/N_iterations;
% c_C = 1.49445*rand;
% c_S = 1.49445*rand;
% c_S = 1.3*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
```

Impulsive

```
function Impulsive()
global P J N_particles mu R1 R2 a_h dv2 dv1 MassRatio_I i_tf c n0 DU TU

i1=0;
delta_inclin= i_tf-i1;

DU=R1; %distance canonical unit
TU=sqrt(DU^3/mu); %time canonical unit
c=0.5*DU/TU; %effective exhaust velocity
```

```
n0=0.16*DU/TU^2; %thurst to mass ratio

a_h=(R1+R2)/2;
energy= -mu/(2*a_h);
vp=sqrt(2*(energy+mu/R1));
va=sqrt(2*(energy+mu/R2));

vc2=sqrt(mu/R2);
vc1=sqrt(mu/R1);
v_inc = 2*vc2*sin(delta_inclin/2);

v2= sqrt(va^2+v_inc^2);

dv2 = sqrt(v2^2 + vc2^2 -2*v2*vc2*cos(delta_inclin));
dv1 = abs(vp-vc1);

MassRatio_I= exp(-(dv2+dv1)/c);
fprintf('Mass Ratio Hohmann= %6.5f \n',MassRatio_I)
end
```

SCEoM1

```
function xdot = SCEoM1(t,x)
global mu zeta0 zeta1 zeta2 zeta3 c n0

xdot= zeros(4,1);
delta= zeta0+zeta1*t+zeta2*t^2+zeta3*t^3; %thrust pointing angle
ToverM=c*n0/(c-n0*t); %thrust to mass ratio

xdot(1) = x(2);
xdot(2) =-1*(mu-x(1)*x(4)^2)/x(1)^2+ToverM*sin(delta);
xdot(3) = x(4)/x(1);
xdot(4) = -1*(x(2)*x(4))/x(1)+ToverM*cos(delta);

xdot = [xdot(1); xdot(2); xdot(3);xdot(4)];
```

SCEoM_pc

```
function xdot = SCEoM_pc(t,x)
global mu zeta0 zeta1 zeta2 zeta3 c n0 dt1 t2 %dt2 dtcoast tf
global alpha0 alpha1 alpha2 alpha3

%xdot= zeros(4,1);Initial conditions arent zero for second thrust

alpha= alpha0+alpha1*(t-t2)+alpha2*(t-t2)^2+alpha3*(t-t2)^3;
```

```
delta= zeta0+zeta1*(t-t2)+zeta2*(t-t2)^2+zeta3*(t-t2)^3; %thrust pointing
angle
ToverM=c*n0/(c-n0*(dt1+t-t2)); %thrust to mass ratio

xdot(1) = x(2);
xdot(2) =x(1)*x(4)^2-
(mu*x(1))/(x(1)^2+x(5)^2)^(3/2)+ToverM*sin(delta)*cos(alpha);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1)+ToverM*cos(delta)*sin(alpha)/x(1);
xdot(5) = x(6);
xdot(6) = -mu*x(5)/((x(1)^2+x(5)^2)^(3/2))+ToverM*sin(alpha);

xdot = [xdot(1); xdot(2); xdot(3); xdot(4); xdot(5); xdot(
```

Appendix B

Future Work: Anomalies in Results

As mentioned in future work, one problem is that in some trial runs the code finds a particle that is within the tolerance for all four equality constraints, but it fails all four in the next iteration. Any way to cut down on the amount of attempts and time needed to find the global minimum would be highly beneficial, so a new feature that will avoid this problem would be helpful. This section displays some of the results being described.

Iteration 11 finds a particle, Particle 26, which is within all the equality constraints, but does not show following iterations. The same result can be seen for Particle 16 during iteration 15.

Iteration : 10

Good alpha2 Value for = 8

GG = 53.48490

dt1 = 1.47516, dtcoast = 28.32745, dt2 = 0.70316, dAng = 172.08264

Mass Ratio = 0.30294

Iteration : 11

Good alpha1 Value for = 26

Good alpha2 Value for = 26

Good alpha3 Value for = 26

Good alpha4 Value for = 26

Good alpha2 Value for = 43

GG = 53.48490

dt1 = 1.47516, dtcoast = 28.32745, dt2 = 0.70316, dAng = 172.08264

Mass Ratio = 0.30294

Iteration : 12

GG = 51.43046

dt1 = 1.31871, dtcoast = 12.18795, dt2 = 0.88300, dAng = 140.98180

Mass Ratio = 0.29545

Iteration : 13

GG = 51.43046

dt1 = 1.31871, dtcoast = 12.18795, dt2 = 0.88300, dAng = 140.98180

Mass Ratio = 0.29545

Iteration : 14

GG = 51.43046

dt1 = 1.31871, dtcoast = 12.18795, dt2 = 0.88300, dAng = 140.98180

Mass Ratio = 0.29545

Iteration : 15

Good alpha1 Value for = 16

Good alpha2 Value for = 16

Good alpha3 Value for = 16

Good alpha4 Value for = 16

GG = 51.43046

dt1 = 1.31871, dtcoast = 12.18795, dt2 = 0.88300, dAng = 140.98180

Mass Ratio = 0.29545

Iteration : 16

Good alpha1 Value for = 40

GG = 51.43046

dt1 = 1.31871, dtcoast = 12.18795, dt2 = 0.88300, dAng = 140.98180

Mass Ratio = 0.29545

Another interesting set of data: the values with good alphas in iteration 37 do not retain themselves in the next ieration, 38.

Iteration : 36

GG = 42.14553

dt1 = 1.35792, dtcoast = 17.00978, dt2 = 1.07984, dAng = 153.52329

Mass Ratio = 0.21992

Iteration : 37

Good alpha4 Value for = 8

Good alpha3 Value for = 9

Good alpha4 Value for = 17

Good alpha3 Value for = 29

Good alpha3 Value for = 44

GG = 41.92105

dt1 = 1.37203, dtcoast = 18.41225, dt2 = 1.10119, dAng = 156.40219

Mass Ratio = 0.20857

Iteration : 38

Good alpha3 Value for = 3

Good alpha3 Value for = 15

Good alpha3 Value for = 25

Good alpha4 Value for = 28

Good alpha3 Value for = 32

Good alpha4 Value for = 32

Good alpha4 Value for = 49

GG = 40.89892

dt1 = 1.36727, dtcoast = 17.94522, dt2 = 1.09451, dAng = 155.48592

Mass Ratio = 0.21223

BIBLIOGRAPHY

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

[2]Curtis, Howard D. *Orbital Mechanics for Engineering Students*. 3rd ed. Butterworth-Heinemann, 2013. Print.

Academic Vita of Peter Flanagan

Pzf5033@psu.edu

The Pennsylvania State University
Bachelor of Science in Aerospace Engineering
Honors: Aerospace Engineering

Thesis Title: Particle Swarm Optimization Applied to Finite Thrust Orbital Transfers between Non-coplanar Circular Orbits
Thesis Supervisor: Robert G. Melton

Work Experience
Date: Summer of 2015
Title: Structural Data Recording Engineering Intern
Validated Structural Test Data for Engines on Test Stands
Pratt & Whitney, West Palm Beach FL
Supervisor: Kenneth Marshal

Date: Summer of 2015
Title: Sensor Applications Engineering Intern
Supported Sensor installations for the PW1000G Engine Family
Pratt & Whitney, Middletown, CT
Supervisor: Dan Abrams

Awards: Dean's List, Sigma Gamma Tau Honors Society

Professional Memberships: AIAA National Member

Publications: AIAA Technical Paper- Undergraduate

Presentations: AIAA Region I Student Conference on Thesis

Activities: Vice President of AIAA Penn State Chapter