A COMBINED GLOBAL AND LOCAL METHODOLOGY FOR LAUNCH VEHICLE TRAJECTORY DESIGN-SPACE EXPLORATION AND OPTIMIZATION

A Thesis
Presented to
The Academic Faculty

by

Michael J. Steffens

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

Georgia Institute of Technology
May 2014

A COMBINED GLOBAL AND LOCAL METHODOLOGY FOR LAUNCH VEHICLE TRAJECTORY DESIGN-SPACE EXPLORATION AND OPTIMIZATION

Approved by:

Professor Dimitri Mavris, Advisor
Aerospace Systems Design Laboratory
Georgia Institute of Technology

Dr. Brad St. Germain
Advanced Concepts Group
SpaceWorks Enterprises, Inc.

Mr. Stephen Edwards
Aerospace Systems Design Laboratory
Georgia Institute of Technology

Date Approved: April 7, 2014

To Jesus Christ, my Lord and Savior

ACKNOWLEDGEMENTS

I take a moment now to express my gratitude to the many people who have helped me along my road here. I have had advisors who have guided me and challenged me, friends and family who have supported me and peers who have encouraged me and critiqued my work. I am grateful to all of them.

To Dr. Mavris, thank you for making my graduate studies possible by welcoming me into ASDL. Thank you for always challenging me to strive for excellence. Thank you for your patient guidance as I learn. It is a privilege to be your student and I look forward to the next thesis.

To Brad St. Germain, and all those at SpaceWorks, thank you for your willingness to teach me and train me. My internship with you provided me with the opportunity to work on relevant problems. Thank you for fostering a welcoming environment. I appreciate all of your insight and feedback on this work.

To Stephen Edwards, thank you for generously giving of your time to meet with me and for helping me work through each problem as it arose. You were a constant source of guidance. I am grateful for your friendship, wisdom, and encouragement.

To my family and friends, thank you for your love and support. Thank you for believing in me and being there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Trajectory optimization is an important part of launch vehicle design and operation. With the high costs of launching payload into orbit, every pound that can be saved increases affordability. One way to save weight in launch vehicle design and operation is by optimizing the ascent trajectory.

Launch vehicle trajectory optimization is a field that has been studied since the 1950's. Originally, analytic solutions were sought because computers were slow and inefficient. With the advent of computers, however, different algorithms were developed for the purpose of trajectory optimization. Computer resources were still limited, and as such the algorithms were limited to local optimization methods, which can get stuck in specific regions of the design space. Local methods for trajectory optimization have been well studied and developed. Computer technology continues to advance, and in recent years global optimization has become available for application to a wide variety of problems, including trajectory optimization.

The aim of this thesis is to create a methodology that applies global optimization to the trajectory optimization problem. Using information from a global search, the optimization design space can be reduced and a much smaller design space can be analyzed using already existing local methods. This allows for areas of interest in the design space to be identified and further studied and helps overcome the fact that many local methods can get stuck in local optima.

The design space included in trajectory optimization is also considered in this thesis. The typical optimization variables are initial conditions and flight control variables. For direct optimization methods, the trajectory phase structure is currently chosen *a priori*. Including trajectory phase structure variables in the optimization

process can yield better solutions.

The methodology and phase structure optimization is demonstrated using an earth-to-orbit trajectory of a Delta IV Medium launch vehicle. Different methods of performing the global search and reducing the design space are compared. Local optimization is performed using the industry standard trajectory optimization tool POST. Finally, methods for varying the trajectory phase structure are presented and the results are compared.

CHAPTER I

INTRODUCTION

The launch vehicle market is driven by both commercial and military space applications. In 2012, as in 2011, there were 80 launches worldwide [3]. While this may not seem like a lot, a cost estimate for a launch is on the order of $100 million. Total revenues in the commercial launch vehicle market in 2012 were estimated at $2.2 billion [3]. The commercial market forecasts predict an overall increase in commercial launch demand over the next 10 years [2]. In addition, military space applications are constantly being launched from various countries. In the United States, the U.S. Space Transportation Policy from 2005 states

> "Access to space through U.S. space transportation capabilities is essential to: (1) place critical United States Government assets and capabilities into space; (2) augment space-based capabilities in a timely manner[...]; and (3) support government and commercial human space flight." [1]

Space launch vehicles are in demand in both commercial and government sectors. They are the "key to space", and therefore strategically important to both businesses and government entities [46].

Launch vehicles are designed, built, and operated with the express purpose of transporting payload from earth's surface to a specified orbit. At the surface of the earth a payload will be at 0 km in altitude and moving less than 1 km/s due to the rotation of the earth. A typical Low Earth Orbit (LEO) mission will transfer a payload from earth's surface to an altitude of 500 km and accelerate it to more than

7 km/s. A simple calculation shows the difference in energy between the states.

$$\Delta E = \Delta KE + \Delta PE = \frac{1}{2}m(v_2^2 - v_1^2) + mg(h_2 - h_1) = \frac{1}{2}m(7000^2) + m(9.81)(500000) \approx 30MJ/kg \tag{1}$$

For every kg of payload about 30 MJ of energy needs to be imparted. This is roughly equivalent to the kinetic energy of a loaded 18-wheeler moving 40 $miles/hour$. This estimate does not take into account any losses incurred during the vehicle launch.

Because of the high acceleration involved in space launch, launch vehicle designers have a measure of launch vehicle performance known as ΔV, or simply put, change in velocity.

$$\Delta V = g_0 \overline{I}_{sp} ln\left(\frac{m_i}{m_f}\right) \tag{2}$$

Where g_0 is the gravitational acceleration at Earth's surface, 9.81 m/s, $\overline{I}_{sp}$ is the specific impulse, and m_i and m_f are initial and final mass respectively [71]. The term $\frac{m_i}{m_f}$ is sometimes referred to as the mass ratio MR. Specific impulse is a measure of the propulsion system efficiency. It quantifies how much thrust is produced for a given amount of mass flow. The higher the specific impulse, the more thrust is produced for a given mass flow rate, and therefore the more efficient the propulsion system. In terms of overall launch vehicle performance, ΔV is linearly proportional to specific impulse.

The natural-log term is a measure of the structural system efficiency. The final mass term is essentially the initial mass minus the propellant-used mass. By minimizing structural mass, the final mass term is minimized, and thereby the natural-log term is maximized. In terms of overall launch vehicle performance, ΔV is exponentially related to the mass ratio. The total mass of a launch vehicle can be broken down into propellant mass, structural mass, and payload mass. Increases in structural mass directly affect the payload weight. In conceptual design a decrease in upper stage inert mass (or structural mass) has a 1 to 1 relationship to payload mass. Every pound

saved is a pound gained for payload mass [71]. Once the vehicle is built, increases in fuel weight required will decrease the payload weight. In general launch vehicle design is driven largely by weight, because any increase in weight has an exponential effect on system performance.

Equation 2 above is sometimes referred to as the ideal ΔV equation, because it measures the maximum total velocity a vehicle can impart if there are no losses during the launch process. However, the ideal velocity change is never achieved. During a launch trajectory there are three types of losses, shown in Equation 3 below.

$$\Delta V_{actual} = \Delta V_{ideal} - \Delta V_{thrust\ vector\ losses} - \Delta V_{drag\ losses} - \Delta V_{gravity\ losses} \tag{3}$$

The three types of losses are categorized as thrust vector losses, drag losses, and gravity losses [71]. Thrust vector losses are due to thrust vector and velocity vector misalignment. Basically, the thrust is being used for a purpose other than to accelerate the vehicle. This can occur due to steering or imprecision in the thrust angle measurements of different engines. In trajectory optimization, the thrust vector losses due to steering are most relevant. Drag losses are due to drag the vehicle experiences as it moves through the atmosphere. Drag is proportional to atmospheric density, which decreases exponentially with altitude. Trajectories that minimize drag losses will gain altitude as quickly as possible to get out of earth's atmosphere. Gravity losses are due to imparting velocity against the acceleration of gravity. When thrust is perpendicular to gravity there are no gravity losses, because no thrust is being used to counteract gravity, and therefore all the thrust is imparting a change in velocity. Obviously, gravity losses are minimized by thrusting horizontally.

The three types of losses are all functions of the vehicle trajectory. Trade-offs exits between minimizing drag losses by gaining altitude and minimizing gravity losses by thrusting horizontally, all the while keeping steering angles small to minimize thrust vector losses. In real world applications, it is impossible to bring all these terms to zero, but the object of trajectory optimization is to find the trajectory that

minimizes the sum of all these losses while not violating any vehicle constraints, such as maximum acceleration or maximum dynamic pressure (used as a measure of aerodynamic loads.)

It is important to note a difference in perspective between launch vehicle design and launch vehicle operation when it comes to trajectory optimization. In launch vehicle design the actual ΔV, sometimes called the required ΔV, is set by the required mission(s). This is the change in velocity that the vehicle needs to be able to impart, including all the losses. The vehicle is then design with a higher ideal ΔV than required, taking into account the ΔV losses incurred during the mission trajectory. In launch vehicle operation the ideal ΔV is fixed; the vehicle has a certain propulsive and structural efficiency. In this case different trajectories can be developed to maximize the actual ΔV given different missions. In either case, the goal is to have the vehicle perform as close to the ideal as possible, and therefore the losses should be minimized. This leads to an important observation: **Optimizing the launch trajectory allows the vehicle to perform closer to its ideal; hence, trajectory optimization is a critical part of both launch vehicle design and operation.**

Allowing a vehicle to perform closer to its ideal is important when one considers typical launch costs. For the Space Shuttle it was estimated that it cost around \$10,000 per *lb* to LEO [58]. The Atlas and Delta launch vehicles operate at a lower cost somewhere between \$3,000 and \$5,000 per *lb* to LEO [58]. The least expensive estimate is for Russian and Ukrainian launch vehicles, at around \$2,000 per *lb* to LEO [58]. From a simplistic perspective, every pound saved on fuel because of trajectory optimization can be translated to additional payload, worth between \$2,000 and \$10,000 per *lb* in orbit. Trajectory optimization is a very important problem and has direct effects on launch vehicle design and operations.

An interesting side note is that as the launch vehicle market grows and demand increases, entrepreneurs are finding ways to cut costs and design, build, and launch

spacecraft in more affordable ways. Elon Musk, for example, at Space Exploration Technologies believes that through manufacturing and process optimization the cost of launching to LEO can be decreased to less than $500 per lb [52]. Even with this drastically reduced cost, trajectory optimization is still required, as each pound of fuel is still significantly costly [65].

The goal of this research is to develop a trajectory optimization method for launch vehicles that is robust, explores all the design space, and does not require a significant amount of human interaction. Chapter 2 will discuss how the current optimization methods are not always robust, generally require cases to be run manually, and do not always take into account the entire design space.

CHAPTER II

LITERATURE REVIEW

2.1 Optimization

2.1.1 General Optimization Problem

The general optimization problem can be formulated mathematically as (as seen in *Multidiscipline Design Optimization*) [68]

Minimize:$F(x)$ objective function

Subject to:

$g_j(\mathbf{X}) \leq 0$	$j = 1, m$	inequality constraints
$h_k(\mathbf{X}) = 0$	$k = 1, l$	equality constraints
$X_i^l \leq X_i \leq X_i^u$	$i = 1, n$	side constraints

Where

$$\mathbf{X} = \begin{Bmatrix} X_1 \\ X_2 \\ X_3 \\ . \\ . \\ . \\ X_n \end{Bmatrix} \tag{4}$$

The design variables are parameters in the user's control that determine the outcome of the objective function. The objective function may be evaluated analytically, numerically, or even experimentally. For the purposes of this discussion, the term performance index is used interchangeably with objective function. Inequality constraints are constraints where some function of the design variables must be less than

zero. Equality constraints are similar, except that the function must equal zero. Side constraints are direct upper and lower bounds on the design variables themselves. In this formulation there are m inequality constraints, l equality constraints, and n side constraints.

An optimal control problem is a type of optimization problem where the goal is to determine "the inputs to a dynamical system that optimize (i.e., minimize or maximize) a specified performance index while satisfying any constraints on the motion of the system."[57] The dynamical system is defined by a set of ordinary differential equations (ODE's). The state of the system at any time is found by solving the set of differential equations given initial conditions and control parameters. The state may be subject to path constraints, which limit how the system can behave, and boundary conditions, which determine the state of the system at the initial and final times. In this case the control is not a single parameter, but a function. Stated mathematically (as formulated in Betts [16])

$$
\begin{aligned}
&\dot{\mathbf{y}} = \mathbf{f}\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] && \text{system equations} \\
&\boldsymbol{\psi}_{0l} \leq \boldsymbol{\psi}\left[\mathbf{y}(t_0), \mathbf{u}(t_0), \mathbf{p}, t_0\right] \leq \boldsymbol{\psi}_{0u} && \text{initial boundary conditions} \\
&\boldsymbol{\psi}_{fl} \leq \boldsymbol{\psi}\left[\mathbf{y}(t_f), \mathbf{u}(t_f), \mathbf{p}, t_f\right] \leq \boldsymbol{\psi}_{fu} && \text{final boundary conditions} \\
&\mathbf{g}_l \leq \boldsymbol{g}\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] \leq \mathbf{g}_u && \text{path constraints} \\
&\mathbf{y}_l \leq \mathbf{y}(t) \leq \mathbf{y}_u \quad \mathbf{u}_l \leq \mathbf{u}(t) \leq \mathbf{u}_u && \text{bounds on state and control variables}
\end{aligned}
\tag{5}
$$

Where $\mathbf{y}(t)$ is the state vector, $\mathbf{u}(t)$ is the control vector, $\mathbf{p}$ is the static parameter vector, and t_0 and t_f are the initial and final times respectively.

For optimal control problems the control vector is a function, rather than a set of design variables. This is known as an infinite dimensional control problem [20]. A non-linear optimization problem or non-linear programming problem (NLP) is one where there are a finite number of input parameters that are used to optimize the system [36]. An infinite dimensional control problem can be approximated by an NLP using a process called transcription [57] [25]. This process will discussed in

more detail in Section 2.2.2.

It is sometimes convenient to divide an optimal control problem into a series of phases. For each of these phases there exist initial and final boundary conditions and path constraints. In addition, there are constraints that link one phase to another. This is used to ensure state vector continuity between phases.

Most real world dynamical systems are complex, and analytic solutions to the system equations are impossible [65]. Because of this the systems of equations are solved numerically [57]. Different methods exist for numerically solving ordinary differential equations, and will be discussed in Section 2.2.1.

2.1.2 Approaches to Optimization

There exist several optimization approaches that have been implemented. The choice of optimization approach is very problem dependent, and the wrong choice can lead to long run-times and poor results. Below are given two broad optimization method categorizations.

2.1.2.1 Global vs. Local Optimization

The global optimum is defined as the set of inputs which yields the best (either maximum or minimum) performance index over the entire design space [67]. In other words, of all the feasible combinations of design variables, the globally optimal set yields the best performance index. A local optimum is a set of inputs which yields the best performance index within a certain subsection of the design space [72] [55]. Figure 1 illustrates the difference between a global and local optimum. An input value of $x \approx -1.1$ for example is a local optimum while $x \approx -0.2$, circled in green, is the global optimum. An important observation is that a global optimum is a local optimum, but not all local optima are the global optimum [68].

The function to be optimized can be categorized as convex or non-convex [11]. Practically speaking, an optimization problem is convex if the local optimum is also

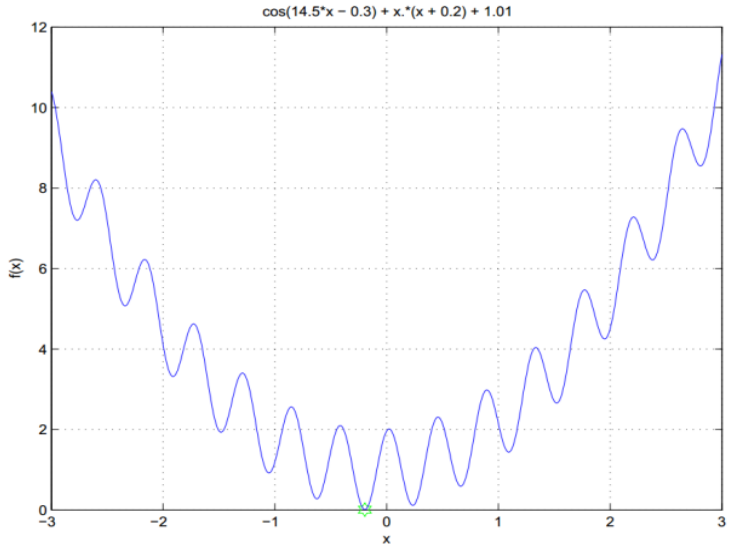


Figure 1: Example of global vs. local optimum [72]

the global optimum [47]. Figure 1, for example, is not a convex problem. If the input variable x were subject to the constrains $-0.3 \leq x \leq 0$, the problem would be convex. Determining if a function is convex or not, however, is not always straightforward, and for many real world problems, convexity cannot be assumed [67]. This means there exists more than one local optimum in the region of interest; these problems are referred to as multimodal problems [67].

Local optimization, sometimes referred to as gradient-based optimization [57], is a method that finds local optimum. As its name implies, it relies on information from the gradient to determine the optimum. The simplest example of a local optimization technique is the line search [68]. In one dimension, a starting point is selected, and steps are taken in a certain direction. At each step the objective function is evaluated. If the objective function improves, another step is taken; if not, a step is taken in the opposite direction. If steps in either direction worsen the objective function, the step size is reduced and the process is repeated. There are many ways to modify direction selection and step size, and this leads to many different algorithms. Some of the more common ones included Steepest Descent Methods, Conjugate Direction Method, Simplex Method, and Sequential Quadratic Programming [68].

Local optimization techniques can be very powerful and are used in many optimization problems. By using gradient information, they are able to hone in on optima without wasting function calls exploring the entire area. For convex problems, local optimization methods are the best option [47]. For non-convex problems, however, local optimizers may get stuck in local minima [8]. For example, referring again to Figure 1, if the starting point was chosen to be $x = -1$, and the step size was small, a local optimizer would output $x \approx -1.1$ as the optimum. This is the main disadvantage of local optimization methods.

Global optimization methods attempt to solve this problem by performing a search not entirely based on gradient information. They are designed to be able to find the global minimum even in highly multimodal problems [55]. A grid search, for example, would partition the design space in some meaningful way and evaluate the objective function at each grid point. For discrete design variables, there exists the option to evaluate every feasible point. For continuous variables this is not possible (although a continuous space can be discretized). Global optimizers have the advantage that they explore the design space more exhaustively than a gradient based optimizer [47]. The opposite side of the coin, however, is that these methods generally require a much higher number of function calls. Recent advances in the area of computing have made global optimization methods more feasible and have lead to a recent surge of research in this area [47]. Another disadvantage of global techniques is the inability to quickly find local optimum. Returning to the grid search example, an algorithm may pick the best performing points from an initial grid search and perform a smaller subsequent grid search around these. As one can imagine the number of function calls using this method can quickly become prohibitively large.

A difference to note between local and global optimization techniques is that generally local optimization works with a single point at a time. The optimizer modifies one candidate solution until it cannot improve further. Global methods

usually work with populations of solutions. At each iteration in a global method anywhere between 10 and 1000 candidate solutions may be evaluated, and information from all of those may be used to determine the candidates for the next population. The output, then, of a local optimization method is a single solution, while the output of a global method can be either a single solution or a family of solutions.

2.1.2.2 Deterministic vs. Stochastic Optimization

Another categorization of optimization methods is based on the use of random numbers. An algorithm is categorized as deterministic if for a given set of inputs, the output is always the same. The term deterministic is defined more formally in Liberti [47], but the above definition will suffice for the purposes of this discussion. Stochastic algorithms, however, use random numbers to generate the output. This means for a given input, the output may vary. At first glance this may seem of little value, but stochastic methods have been used extensively in optimization. A more in depth discussion of how stochastic methods are used in trajectory optimization is included in Section 2.2.4. An interesting note is that computer algorithms are inherently deterministic. Therefore, to generate random numbers, pseudo-random number generators are used to simulate randomness [47].

One matter of importance regarding stochastic algorithms is the issue of convergence. For a global optimization method, convergence refers to how well the method finds global optimum [67]. Stochastic algorithms are based on probability, and therefore convergence is not guaranteed for anything less than an infinite number of cases. This is obviously not feasible, so these algorithms must be generated in such a way as to maximize the probability of convergence. One way to do this is simply to run a specific case multiple times [54].

These two categorizations, global vs. local and deterministic vs. stochastic, are independent and can be combined in any way. A global method, for example, can

be either deterministic or stochastic, and there exist both local and global stochastic methods. However, most local methods are deterministic, and global methods are generally stochastic [47]. This is because local methods inherently have a path to follow based on gradient information, which is defined by the system equations, not the optimization method. Global methods, while trying to explore the whole space, attempt to spread out and look at areas that may not seem promising, and random numbers can be used to achieve this.

2.2 Trajectory Optimization

Trajectory optimization is an infinite dimensional control problem. Recall from Section 2.1.1 that the solution to an infinite dimensional control problem is a function. There are several ways to approach these infinite dimensional trajectory optimization problems. As with most real world optimal control problems, trajectory problems are solved numerically [57]. In literature, the terms trajectory optimization and optimal control problem are used interchangeably. However, there exists a practical distinction between the two that arises from the way trajectory problems are optimized. If the inputs to the system are static parameters, the appropriate term is trajectory optimization. The term optimal control is used to refer to problems where the inputs to the system are themselves functions [57] [16]. At first glance this seems contradictory. After all, trajectory problems are optimal control problems. However, the most common approach to solving trajectory problems is to divide the problem into phases [57] and approximate the controls as constants or polynomials [37], which can be expressed as a set of parameters. This reduces the optimal control problem to an NLP and is known as a direct method. Another method is the indirect method. Both these methods will be discussed in this section.

As mentioned before there are several methods that have been successfully implemented to optimize trajectories. Trajectory optimization problems are categorized

both by how they approach the optimization problem and how they solve the dynamic system of equations. In this section an overview of the current trajectory optimization techniques and the mathematical tools required is provided. Because complicated optimal control problems require numerical simulation, optimization methods can be categorized by numerical simulation method. For trajectory optimization there are two main methods used for solving the ODE's: shooting and collocation.

2.2.1 Numerical Integration Methods

Regardless of how the trajectory optimization problem is solved, whether via direct or indirect methods or local or global methods, the differential equations must be solved [37]. As stated earlier, analytic solutions for the differential equations of problems of this complexity do not exist, so numerical methods must be employed [65]. In trajectory optimization, the first of two main methods is called shooting, or time-marching.

2.2.1.1 Shooting Method

Shooting calculates the current state based on information from either current or previous state information. Essentially, at each time step the system equations are calculated, and the resulting derivatives are used to update the state to the next time step. There are several techniques on how the derivatives are used to update the state. Euler methods are shown in equation 6

$$x_{k+1} = x_k + h_k \left[\theta \mathbf{f}_k + (1-\theta)\mathbf{f}_{k+1}\right] \tag{6}$$

Where $\mathbf{f}_k = \dot{\mathbf{x}}$ and h_k is the time step. When θ is 1, the method is called Euler forward, because the next state is dependent entirely on the information from the previous state. This type of numerical integration is called explicit integration [13]. When θ is 0, the Euler backward method is used [12]. In this case the next step is dependent on the previous state values but derivatives from the next state. These

methods are called implicit integration methods because the state x_{k+1} is on both sides of the equation [13]. Because of this, the equations must be solved iteratively. In general, explicit methods are easier to implement and more computationally efficient, but not as accurate as implicit methods.

Euler methods are the simplest form of shooting methods [66]. Probably the most common numerical integration method, however, is an explicit fourth order Runge-Kutta method [57]. This method is shown in Equation 7 below.

$$\begin{aligned}
\mathbf{k}_1 &= h_i \mathbf{f}(\mathbf{x}_i, t_i) \\
\mathbf{k}_2 &= h_i \mathbf{f}(\mathbf{x}_i + \frac{h_i}{2}\mathbf{k}_1, t_i + \frac{h_i}{2}) \\
\mathbf{k}_3 &= h_i \mathbf{f}(\mathbf{x}_i + \frac{h_i}{2}\mathbf{k}_2, t_i + \frac{h_i}{2}) \\
\mathbf{k}_4 &= h_i \mathbf{f}(\mathbf{x}_i + h_i\mathbf{k}_3, t_i + h_i) \\
\mathbf{x}_{i+1} &= \mathbf{x}_i + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned} \tag{7}$$

The two methods discussed thus far, Euler and Runge-Kutta, are single-step methods, because only one point is used to compute the second point (even though implicit methods require the current and previous point). There are several other types of single-step methods, such as Heun and Taylor methods [49]. Another class of numerical methods use several of the previous steps (once the algorithm has been started) to compute the next step. These are known as Predictor-Corrector methods. Predictor-Corrector methods are generally more complex, but can be more accurate. One example used in spaceflight trajectory optimization is the Adams-Bashforth-Moulton method [14]. For this study, single-step methods are used because those are the methods implemented in current trajectory optimization tools, as will be discussed in Section 2.2.3. The reader is referred to Mathews and Atkinson for a more in depth discussion on these numerical methods [13] [49].

2.2.1.2 Collocation Method

In literature collocation is sometimes referred to as transcription [16] [38]. Note, the word transcription is used differently when speaking in the context of direct optimization methods, discussed in Section 2.2.3. For the purposes of this discussion, transcription will be used only in the context of direct methods, and collocation will be used to refer to the method of numerically solving differential equations.

Collocation employs an interpolating function to approximate the state of the system. Usually the interpolating function is a polynomial. At collocation nodes, constraints are used to compare the derivative of the approximating function to the solution of the system of equations at that point. These constraints are called defect constraints and are shown in equation 8.

$$\xi = \mathbf{X}(t_j) - \mathbf{f}(\mathbf{x}(t_j), t_j) \tag{8}$$

Figure 2 below illustrates the collocation method. Frank [31] summarizes the collocation method as construction of "a polynomial that passes through y_0 and agrees with ODE at s nodes on $[t_0, t_1]$. Then [...] let the numerical solution be the value of this polynomial t_1." In this context the time step between t_0 and t_1 is broken up into s nodes referred to as $c_1, c_2, ..., c_n$. The red lines in Figure 2 represent the slope of the polynomial, which is compared to the numerical solution of the system equations. Because of the way the collocation method solves differential equations, namely solving for all the variables at once, it is considered an implicit method.

Now that the solution method to a set of ODE's has been discussed, different local optimization techniques will be explored.

2.2.2 Local Optimization Methods

Before the 1990's local trajectory optimization methods were the only feasible methods due to lack of the computational resources to make global optimization a realistic option [16]. Local methods of numerical optimal control fall into two distinct

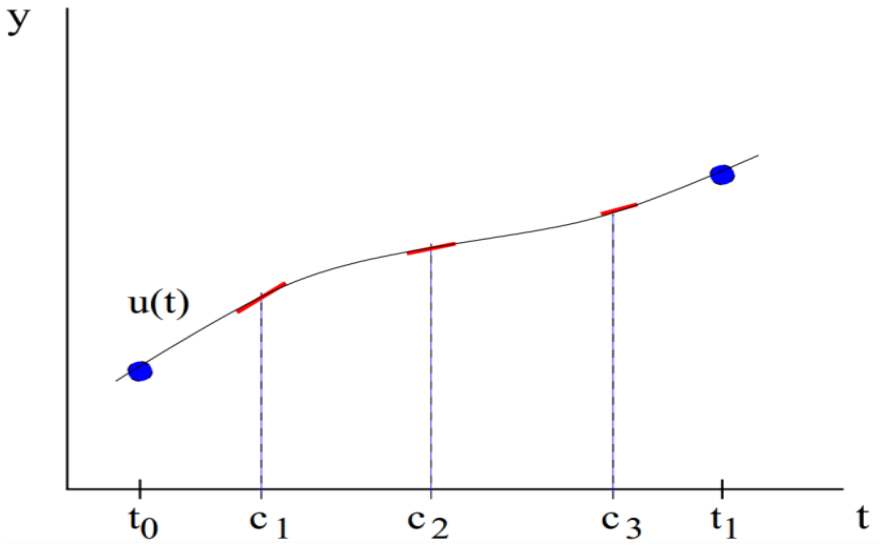


Figure 2: Collocation method [31]

branches: direct and indirect methods.

2.2.2.1 Direct Methods

Direct optimization methods solve the infinite dimensional optimal control problem by converting it into a finite dimensional non-linear programming problem. Direct methods break up the control function in a process called transcription [57]. Note that the term transcription here is different from transcription in the context of collocation. In this paper, the term transcription will only be used in the context of direct optimization methods. The transcription process involves the discretization of the problem into phases. A phase starts and ends at phase events or nodes. Equation 9 shows how the control function could be discretized [65]. In this case there are n

phases.

$$u = u(t) = f(t) \rightarrow u = u(t) = \begin{cases} u_1, & t_1^i \leq t < t_1^f \\ u_2, & t_2^i \leq t < t_2^f \\ & . \\ & . \\ & . \\ u_n, & t_n^i \leq t < t_n^f \end{cases} \tag{9}$$

The parameters $u_1, ..., u_n$ may be constants themselves or they may be functions defined a finite set of parameters. For example, $u_2 = u_{2,0} + t \times u_{2,1}$. In this case the control u_2 is described by a constant term $u_{2,0}$ and a rate $u_{2,1}$. In general, any function can be employed. The parameters required to define the function become the design variables in the optimization problem. Figure 3 is a pictorial depiction of trajectory and control discretization [28]. In both Equation 9 and Figure 3 the trajectory was discretized using time. However, this is not required. A trajectory may be discretized using any variable and even different variables in the same trajectory. Complications may arise when using different variables, and in general time is a common variable to use.

When a problem is transcribed, the resulting non-linear programming problem inherently has fewer degrees of freedom than the optimal control problem. In fact, this is the very reason problems are transcribed. However, by doing this, solutions become sub-optimal [43]. The control function is being approximated by some function, and hence this approximation leads to sub-optimal solutions to the optimal control problem, even if the non-linear programming problem solution is itself optimal.

Direct methods have the advantage of being relatively robust (compared to other local methods) and relatively simple to implement [25]. However, they are less accurate [63] than the second branch of local optimization methods: indirect methods.

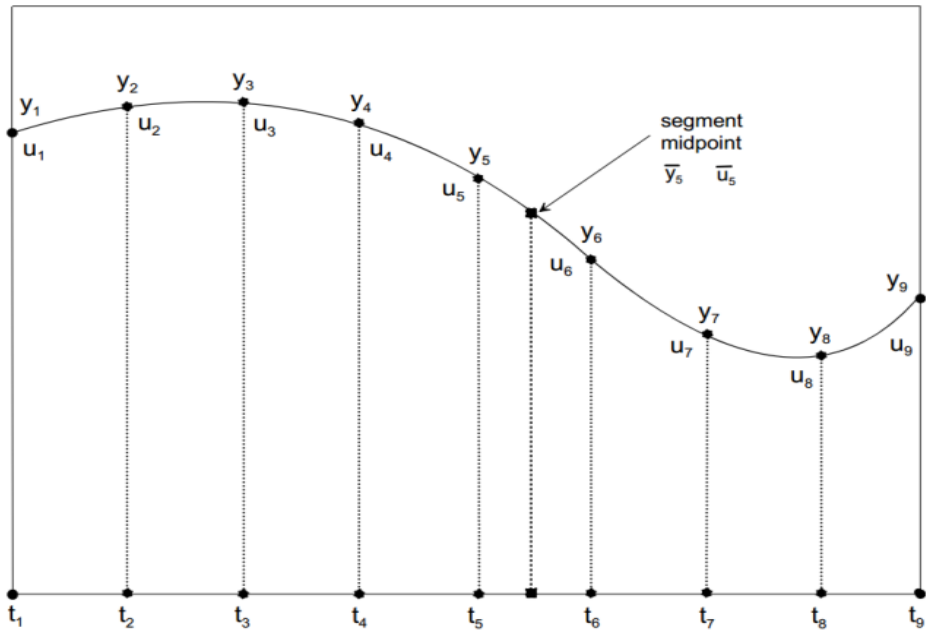


Figure 3: Trajectory and control discretization [28]

2.2.2.2 Indirect Methods

Indirect methods solve the optimal control problem using calculus of variations. A solution is obtained by deriving necessary conditions. An augmented performance index is created using Lagrange multipliers or costates to include the constraints [43]. This results in a boundary value problem [65]. An optimal control function is derived based on the dynamic system and the constraints. Indirect methods solve for a control at each point in time by defining a function for the control that can be solved at any point. Guidance algorithms are derived by solving optimal control problems indirectly [33].

Indirect methods are very powerful, and yield very accurate results. However, deriving the necessary conditions for complex systems can be very difficult. In addition, these necessary conditions are unique to each problem, so developing a tool for general trajectory optimization is challenging to say the least. Finally, because the costates are included as part of the optimization process, the number of optimization variables increases. Initial guesses for costate values are difficult to obtain as well,

because they have no physical meaning. One solution is to randomly guess different initial values for the costates until a solution is found [33]. However, this can lead to long run-times. Indirect methods have been studied significantly, and the general consensus on these methods is summed up in the following quote, as referenced in Betts [16]:

> "The main difficulty with these methods is *getting started*; i.e., finding a first estimate of the unspecified conditions at one end that produces a solution reasonably close to the specified conditions at the other end. The reason for this peculiar difficulty is the extremal solutions are often *very sensitive* to small changes in the unspecified boundary conditions... Since the system equations and the Euler-Lagrange equations are coupled together, it is not unusual for the numerical integration, with poorly guessed initial conditions, to produce "wild" trajectories in the state space. These trajectories may be so wild that values of $x(t)$ and/or $\lambda(t)$ exceed the numerical range of the computer!" [7]

Because of this, indirect methods have not been widely implemented in many general trajectory optimization problems [16] [25].

2.2.3 Local Trajectory Optimization

In local trajectory optimization the combination of the optimization methods and numerical integration methods leads to four main methods. A fifth method is also considered, even though it is only an extension of two of the four main methods. Table 1 shows the four main methods.

A brief discussion of each method is included to investigate benefits and challenges of each method as well as discuss any tools have have implemented these methods.

Table 1: Summary table of local optimization methods

		Optimization Method	
		Direct	**Indirect**
Differential Eqn. Method	**Shooting**	Direct Shooting: • approximates optimal control problem as NLP • employs explicit or implicit integration	Indirect Shooting: • solves necessary conditions to obtain control values • employs explicit or implicit integration
	Collocation	Direct Collocation: • approximates optimal control problem as NLP • approximates system states with polynomials	Indirect Collocation: • solves necessary conditions to obtain control values • approximates system states with polynomials

2.2.3.1 Direct Shooting

Direct shooting is the easiest of these methods to understand and visualize. As a direct method, the control function is represented by a finite set of parameters [25]. A cost function, including path and final constraints, is evaluated by numerically integrating the equations of motion, given initial and ending conditions. The control parameters are then modified based on gradient information to improve the cost function [57]. Optimization algorithms are techniques on how to modify the control parameters, and there are many different algorithms that exist [68].

Direct shooting methods work well when the control function can be approximated by a small number of parameters. Because gradients are calculated for each parameter, as the number of parameters increases the process becomes computationally expensive. Gradient calculation can be another problem for the direct shooting method. Numerical issues can lead to inaccurate gradient calculations. In addition,

small changes in control parameters can lead to extremely non-linear behavior in system constraints. This makes it very difficult for an optimizer to solve the problem [16].

Despite all these difficulties, direct shooting is the most common method in trajectory optimization. Program to Optimize Simulated Trajectories (POST) is a tool developed by NASA Langley Research Center and Lockheed Martin Astronautics in the 1970's. It is used to calculate trajectories for air-breathing or rocket ascent and reentry in arbitrary environments [19]. POST has three optimization algorithms built in. The first two are the un-accelerated and accelerated projected gradient method (PGA) [56]. These only use first order gradient information. The third algorithm is an optimization package NPSOL developed by the Systems Optimization Lab at Stanford University [56]. NPSOL employs second order gradient information, which can be more accurate, but generally takes longer. For more information about the formulation of POST the reader is referred to the POST Formulation Manual [18] or to Brauer [19].

Since the original version, there have been several upgrades to POST. Some of the notable ones include adding 6 degrees of freedom as well as the capability to simulate multiple vehicles at once.

2.2.3.2 Indirect Shooting

Like direct shooting, indirect shooting uses numerical integration, either explicit or implicit, to solve the set of ODE's. However, instead of discretizing the control function, the necessary optimality conditions are derived based on a cost function augmented with Lagrange multipliers and constraints [33]. This leads to a set of optimization variables including simulation time and Lagrange multipliers, or costates. If path constraints are included, they are dealt with by discretizing the trajectory into phases based on whether they are constrained or unconstrainted and solving the

individual phases [16]. This will increase the number of design variables.

There are several complications that arise when using indirect shooting. As with all indirect methods, the necessary conditions must be derived. This can become very complicated. A program like POST allows the user to select from multiple reference frames and different environment models. Deriving the necessary conditions while allowing different reference frames and environment models would be an arduous task. Because of that most indirect shooting tools are considered somewhat inflexible [37], and only useful for a small set of specific problems. There is work being done to overcome these challenges [16].

2.2.3.3 Direct Collocation

Collocation methods were first implemented in indirect optimization, but then applied to direct methods to remove the requirement of deriving the necessary conditions. In direct collocation the set of ODE's is replaced by a set of defect constraints at grid points. When formulating a problem in this way the number of variables increases dramatically. The set of optimization variables includes state and control variables at each grid point as well as initial and final conditions. A problem may have on the order of thousands of optimization variables. Calculating the required matrices in the NLP problem proves costly. However, for direct collocation problems, about 99% of the entries of these required matrices are 0, and therefore algorithms are used to take advantage of the matrix sparsity to reduce computational costs [16].

When set up correctly, direct collocation can be a very powerful method for optimizing trajectories. For this approach to be efficient, however, matrix sparsity must be taken advantage of, and this can be difficult to implement. The method of direct collocation has been implemented in a tool called Optimal Trajectories by Implicit Simulation (OTIS). OTIS was developed in the mid 1980's by NASA Glenn Research Center and The Boeing Corporation. OTIS was designed to optimize trajectories

for many types of vehicles, including launch vehicles, satellites, and aircraft. While OTIS has an option to use the shooting method to solve the ODE's its power lies in the collocation method. The goal of OTIS was to produce a general purpose trajectory simulation and optimization analysis tool. Like POST, OTIS is integrated with optimizer packages. One of the original ones was Boeing's Chebyshev Trajectory Optimization Program (CTOP) [37]. In the latest version a more powerful optimizer is used: SNOPT 7. SNOPT 7 is a sparse non-linear programming optimizer developed at the Systems Optimization Lab at Stanford University. It is designed to take advantage of the matrix sparsity of these type of problems [51]. A more complete description of OTIS is given by Hargraves [37].

Currently OTIS and POST, discussed previously, are the two main trajectory optimization tools used in industry. Opinions regarding which tool provides better results are generally based on which tool the user is more familiar with. The underlying physics in both tools, however, is the same. It has been shown, that there is little numerical difference in the results calculated by POST and the results calculated by OTIS for a given problem [53].

It is interesting to observe that both major tools used in industry apply direct optimization techniques. This is no coincidence. The inherent issues with indirect optimization make it difficult to easily apply to general trajectory problems. Therefore, the tools that have been adopted by industry as standard all employ direct techniques. For this reason, in this thesis, only direct tools were considered to generate results.

2.2.3.4 Indirect Collocation

Indirect collocation was developed to solve necessary conditions for boundary value problems using a different numerical solution technique. Indirect collocation methods have been used in many different applications. However, these methods suffer from

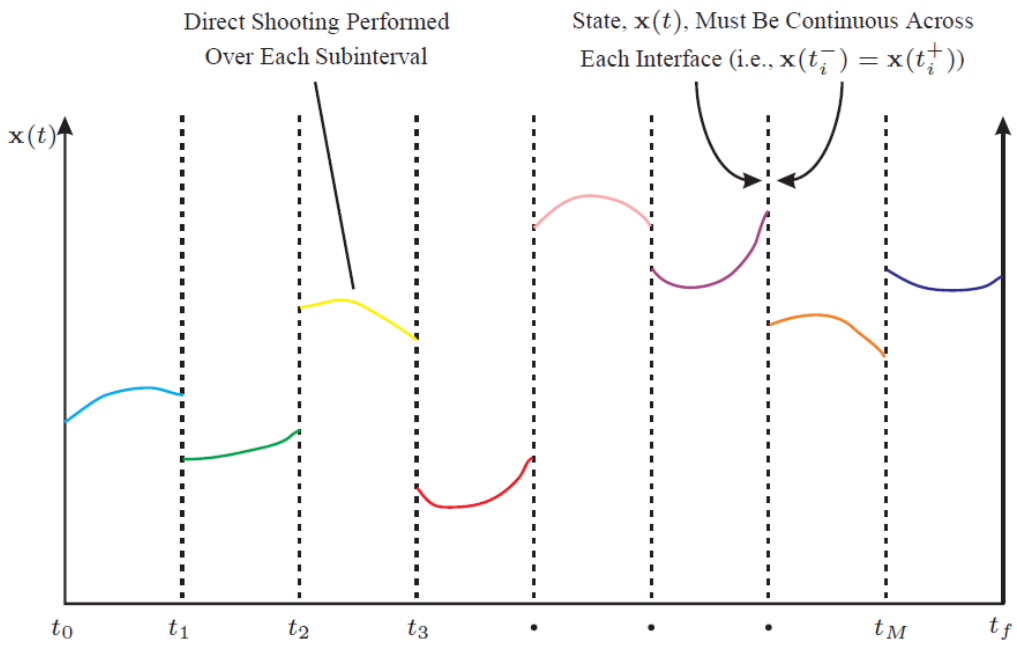


Figure 4: Schematic of direct multiple-shooting method [57]

many of the same problems as indirect shooting methods, and therefore have not been implemented for general purpose trajectory tools.

2.2.3.5 Multiple Shooting

When shooting methods are employed, small changes in control variables early in the trajectory have a large effect on the trajectory's ending state. This is due to long simulation times. A solution to this is to employ a multiple shooting method. Essentially, the idea is to perform a shooting based optimization, either direct or indirect, for segments of the total trajectory, and enforce continuity constraints at the phase boundaries. It is similar to the collocation method, in that the equivalent to a defect constraint is introduced to ensure the simulation is physically feasible. Figure 4 shows the direct multiple shooting method as an example.

Multiple shooting does increase the number of variables needed to solve the system. However, even with this increase in number of variables, multiple shooting methods can be an improvement to single shooting methods. In general, each phase

of a multiple shooting method will have the disadvantages associated with the single shooting method employed, either direct or indirect [57].

2.2.3.6 Local Methods Summary

In summary, local methods are powerful tools that are widely used in trajectory optimization and when computational resources are limited, they are the best option. However, there are certain drawbacks: namely the requirement for a good initial guess (especially important for indirect methods) which comes from a user and the fact that local methods can get stuck in local minima [8] [65]. This information leads to an important observation: **Using local methods alone for trajectory optimization requires significant user input and does not lead to a robust optimization process.**

2.2.4 Global Methods

Global methods have not always been popular in trajectory optimization. In a paper published in 1998, a detailed review of the then current trajectory optimization techniques referred to global methods as simply not worth it. In a discussion about genetic algorithms specifically, the author states

> "Unfortunately, because they do not exploit gradient information, they are not computationally competitive with the methods in Sec. IV. [local methods]." [16]

In the late 1990's, however, there was a significant increase in state-of-the-art computational performance. This led to a surge of research in the area of global optimization methods [47]. Global techniques have the advantage of being able to fully explore the design space without getting caught in local minima.

When global techniques are employed, they are usually coupled with a local optimizer to fine tune candidate solutions. The global technique is used to exhaustively

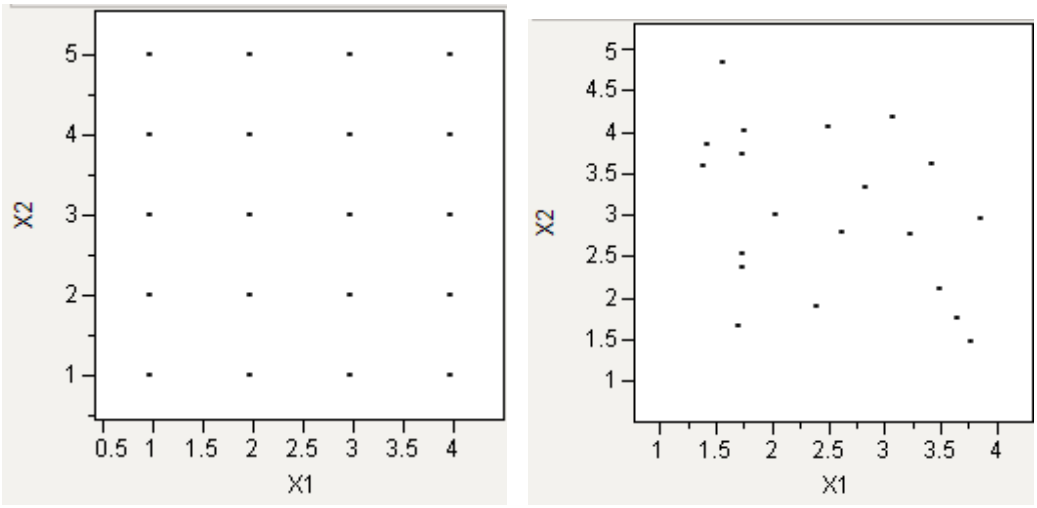


Figure 5: Grid vs. random search

explore the design space and find a family of candidate solutions. Once solutions of interest are found, the local technique is used to refine the solutions [47] [21]. The integration of the local and global methods can take many forms [8]. The simplest way may be to run the global method, then evaluate top candidate solutions using the local optimizer. More sophisticated approaches, however, may yield better results.

Grid searches and random searches are two simple global methods. Whether they are optimization methods or search methods is a matter of debate, and will not be explored here. For the purposes of this discussion they will be referred to as optimization methods. Grid searches allocate a certain number of points to each variable in the design space and evaluate the objective function at each combination of points. Random searches randomly select points inside the design space to evaluate the objective function at. Figure 5 below illustrates grid vs. random search.

As far as the author can tell, only one tool, QuickShot, has been developed specifically for trajectory optimization using global methods. Global methods have been employed for trajectory optimization problems, but no tool developed. QuickShot is a tool that has been developed by SpaceWorks Enterprises, Inc. with the purpose of decreasing the number of trajectory assumptions, decreasing the need for a good

initial guess, and avoiding the need for an expert user in the loop [34]. Global methods provide all these advantages at the cost of increased computational requirements. QuickShot was validated against POST to show similarity of optimized trajectories. The global search methods employed in QuickShot are the grid and random searches. The best results from the global search are then input into a local search to find the local optimum. A schematic of QuickShot is shown in Figure 6.

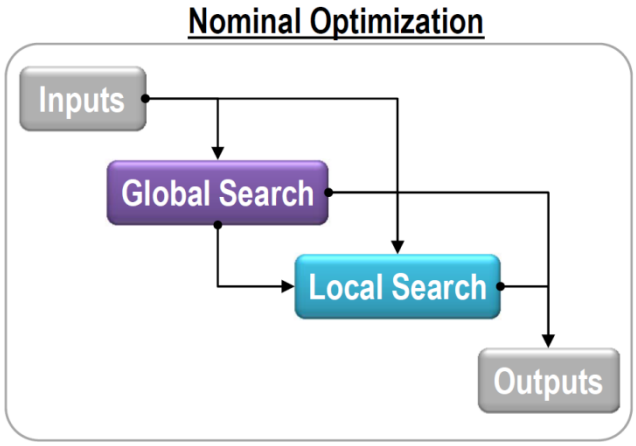


Figure 6: QuickShot optimization method [34]

In the following sections an overview of some of the most common global methods is given. This is a relatively new field in trajectory optimization, and many methods are being employed and updated. This discussion by no means includes all the options, but briefly describes some of the more well-known methods. One note before continuing is that global methods can use either collocation or shooting to numerically integrate the trajectories.

2.2.4.1 Genetic Algorithm

Genetic Algorithm (GA's) were developed to model the theory of evolution. They are inherently designed to work with discrete design variables, but can be easily modified to accommodate continuous variables [68]. They way GA's work is by representing a point in the design space by a binary string. Operations, such as reproduction, crossover, and mutation, are performed to modify the binary string, which leads to

a new point in the design space. The operations performed are designed to only let the best solutions survive (survival of the fittest).

A basic GA search is performed in the following way. A number of random points are chosen to initialize the algorithm. The set of current solutions is known as the population, and a single solution is called a member. A new population is generated every iteration. The series of populations are termed generations. The performance index is evaluated for all members. Members of the population are randomly chosen to perform crossover (exchange binary information) based on the performance index. The better the performance index, the higher the probability that a member will be selected for crossover. Additionally, a mutation process is performed by randomly flipping some of the binary bits [68]. There are many ways to introduce randomness into a process like this. The idea is to have a higher probability of converging on the best solution, but still have a chance of exploring other parts of the design space.

GA's have been applied to many launch vehicle design and launch vehicle trajectory design problems [21] [60] [15].

2.2.4.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is the inherently continuous version of GA's [68]. They are modeled after a flock of birds swarming around food. Each member of the population is represented by a position (the design variables) and a velocity. At every iteration in the algorithm, each member is updated in the following way:

$$X_k^l = X_{k-1}^l + v_{k-1}^l \delta t \tag{10}$$

Where X_k^l is the l member of the k population, and v_{k-1}^l is its corresponding velocity. δt is an arbitrary step size parameter. The velocity is updated each iteration as well based on an inertia parameter (basically how much is the new velocity dependent on the previous velocity), and two trust parameters that weight the best solution among all the members at the current iteration and the best solution of the member

in question throughout all the iterations [68].

A PSO search is performed by initializing a population and updating the position of each member according to Equation 10. After evaluating all the performance indices, the velocity of each member is updated, and the algorithm is repeated [68].

A PSO algorithm was implemented in a study for Reusable Launch Vehicle trajectory optimization [22]. The study concluded that PSO algorithms work will even with small population sizes, which cuts down on computational costs.

2.2.4.3 Differential Evolution

Differential evolution (DE) is based off GA's but like PSO, is designed for continuous variables. It is arguably one of the most powerful stochastic global optimizers currently used [26]. Like GA's a population is seeded to start the optimization process. Each member is represented by a vector of design variables. To understand the algorithm some terminology is required. A current, or parent vector, is updated into a trial vector (in the new generation) via a donor vector. The donor vector is created by using the parent vector and a weighting from two other vectors. Sometimes both other vectors are selected at random, while other times the best vector from the current generation is included with another random vector. The donor vector can be created on a vector by vector basis (i.e. for each donor vector, information from two other vectors are used), or on a variable by variable basis (i.e. each design variable in each donor vector uses information from two other vectors). In addition, the weighting between the two vectors can itself be a random number, again on a vector by vector or variable by variable basis. At this point the trial vector is created by randomly selecting the design variables from either the donor or parent vector. After all this the trail vector performance is compared to the parent vector performance and the best solution is kept. There are several variations of DE's and the parameters that control how the algorithm behaves can themselves be design variables in an

optimization process [26].

DE algorithms have been applied to aerospace problems. Specifically, it was applied to the ascent launch trajectory optimization of a hypersonic vehicle, and was shown to outperform the equivalent NLP problem [35].

2.2.4.4 Global Methods Summary

There are several challenges associated with global searches. The most obvious is the computational requirements. Global methods are able to explore large design spaces, but the number of cases required can lead to infeasible run-times, even with state-of-the-art hardware. Another big challenge is termination criteria; i.e. when to stop the algorithm. There are several ways to terminate a global algorithm. The simplest is a hard limit on number of generations. More sophisticated methods involve looking at how much improvement has occurred over a certain amount of time, or how clustered the population is in the design space, or how much the current "best" point moves vs. how much the performance index changes. Because relevant global algorithms are stochastic, convergence (recall from Section 2.1.2.2 that convergence refers to finding the global minimum) is not guaranteed without an infinite number of cases [47]. Finding termination criteria that leads to good solutions without excessive run-time is challenging. Finally, because relevant global algorithms are stochastic, solving the same problem twice will yield different results. For each problem it is important to determine how to best allocate computational resources, whether to run one search for a long time, or multiple shorter searches. An important observation can be determined from this research on global methods: **Using global methods alone for trajectory optimization will allow for full design space exploration, but may not lead to the most efficient use of computational resources.**

2.2.5 Phase Discretization Methods

As discussed in Section 2.2.3, the two industry standard trajectory tools use direct methods. In fact, any general purpose trajectory optimization tool based on current methods will employ direct methods. Direct methods rely on a discretization of the trajectory, in most cases based on time. This temporal discretization is done *a priori* [10]. In other words, when the problem is set up, a control discretization is chosen, often without analysis. In optimal control problems across several industries it has been shown that the choice of discretization plays a key role in the computational requirements as well as accuracy of the solution method [9].

Adaptive grid methods are used in transcribed optimal control problems to find the best grid, or discretization, for solving the resulting NLP problem. When using adaptive grid methods there is a tradeoff between computational effort and improvement of the solution [9]. If only solution accuracy was considered, an infinite number of grid points would be used, and the NLP problem would approach the optimal control problem.

Three general types of adaptive grid methods exist. In h-refinement extra nodes are added at strategic points to increase accuracy in critical areas. In p-refinement a different numerical method is used to solve the equations of motion in critical areas. Finally in r-refinement a fixed number of nodes are moved around to find the best distribution. Historically, grid methods have been applied to node distribution in the spatial domain [10]. In trajectory optimization, however, most node distribution is in the time domain.

In 2006 a paper was published that stated that "utilizing adaptive node distribution [...] and on-line trajectory optimization has not been considered elsewhere [10]." This is an important observation. The optimal control method is approximated as a NLP problem. If the approximation is not accurate, it can lead to poor results. Finding a good way to include the required temporal discretization of direct method

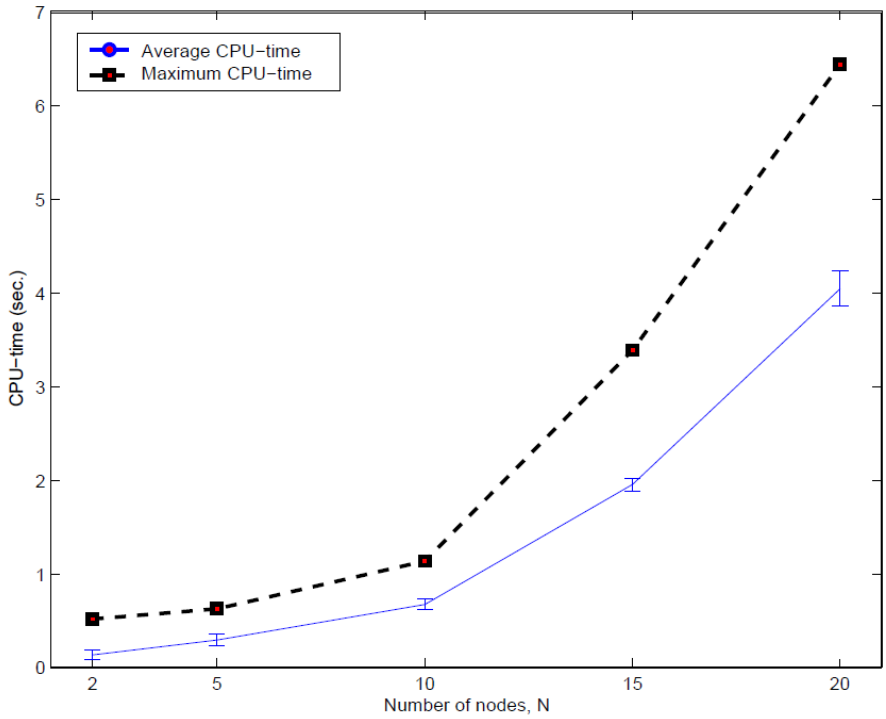


Figure 7: Computational cost vs number of nodes [10]

trajectory problems in the optimization process could yield improved solutions. There is a tradeoff between number of NLP design variables and computational time. Figure 7 shows how run-time increases for a specific problem as the number of nodes increases.

There can be seen a significant increase in computational time vs. number of nodes, and as the number of nodes increases, the rate at which the CPU time increases as well, leading to what appears to be an exponential growth trend. However, the question remains, for example, if it is better to include 10 design variables with a fixed grid placement or 5 design variables and 5 phases, but include the phase event placement parameters as design variables. The number of variables would be the same, so run-times would be similar (because the problems are inherently different, the run-times will not necessarily match). Significant improvement may be seen by including a combination of traditional variables and grid placement variables.

The effect of adaptive node placement is shown in Figure 8. The control error

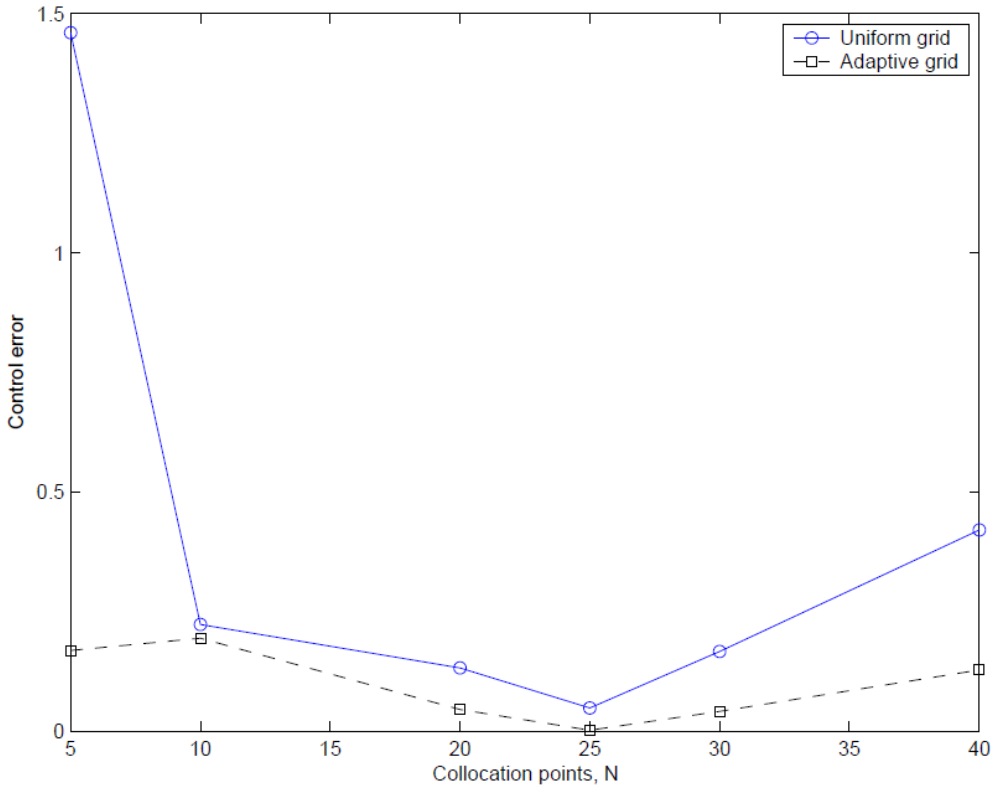


Figure 8: Control error for different grid methods [9]

is plotted against the number of nodes used for adaptive and uniform methods. It is interesting to note that increasing the number of nodes does not always increase the accuracy of the result. The source noted that this was because a local optimizer was used, and global optimality was not guaranteed [68]. In theory, increasing the number of nodes will always decrease the error of the NLP solution when compared to the optimal control solution. It becomes a question, however, of computational effort, and a small improvement may not be worth the added expense.

It is concluded then, that node placement should be investigated in trajectory optimization, and for a general purpose trajectory optimization tool, "node distribution should be a part of the optimization process." [10] The results shown here were published in 2006 [10] [9] and no other references were found since then. As far as the author can tell, there have been no other publications dealing with control node distribution in trajectory optimization, and the results here

have not been referenced in any more recent work.

An important point is made here that significant effort has gone into determining how to place the nodes for numerical integration for the solution of ODE's [61], both for optimal control problems in general [23] [29] [44] and specifically for trajectory optimization problems [59] [36]. The work cited here deals with the numerical integration. This thesis is dealing with the node placement required in the transcription of the optimal control problem to an NLP that is currently not part of the optimization process.

CHAPTER III

METHODOLOGY DEVELOPMENT AND HYPOTHESES

In this chapter a case study is performed to exemplify some of the observations from Chapter 2. In addition, based on the observations in Chapter 2 and the results of the case study in Section 3.1, several research questions and corresponding hypotheses are formulated. Finally, experiments are developed to answer the research questions. It is helpful to keep in mind the overall objective of this research when reading the following sections. Recall the goal of this research is to develop a trajectory optimization method for launch vehicles that is robust, explores all the design space, and does not require a significant amount of human interaction.

3.1 Case Study

Several of the observations made throughout this discussion can be easily seen in a sample trajectory optimization problem set up in POST. As stated earlier, POST and OTIS are very similar numerically. So while this case study was not repeated in OTIS, similar behavior can be expected [53].

The case study was taken from a rocket-back trajectory study [32]. In a rocket-back trajectory the concept is for a launch vehicle to fly to some staging point, release a second stage, and then return to the launch pad via a rocket burn and aerodynamic maneuvers. For this simplified case study, only the launch to staging point was considered. Figure 9 shows the entire rocket-back trajectory, with the section in red being the case study considered.

The vertical launch simulation was initiated at Cape Canaveral launch site. The vehicle gross weight was $820,000\ lb$ plus a payload weight of $223,731\ lb$, and sea level thrust and ISP were approximately $1.26 \times 10^6\ lb$ and $305\ s$ respectively. Propulsion

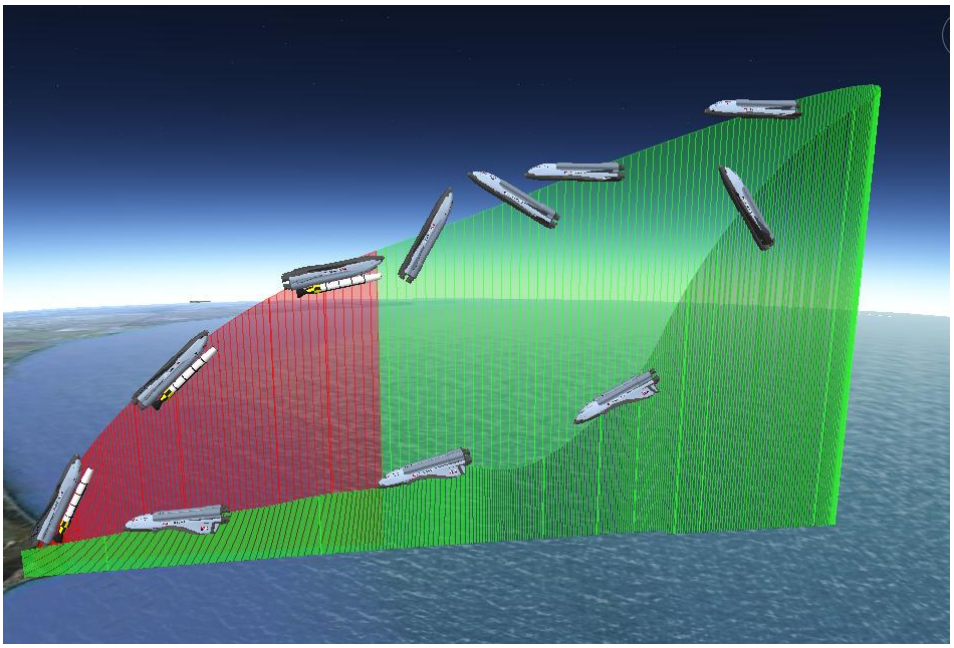


Figure 9: Case study trajectory [32]

and aerodynamic data was included to simulate motion through the atmosphere. Terminal conditions were a velocity of 5500 ft/s, a flight path angle of 20°, and a dynamic pressure of 25 psf. Several constraints on variables such as wing loading and dynamic pressure were included. This optimal control problem was transcribed into 10 phases based on time and one based on velocity. A control variable was assigned to each phase but the first. The control variables in this simulation were pitch rates. Figure 10 shows the trajectory broken down into phases. Each of the brackets represents a phase.

Several experiments were done to illustrate some of the challenges of trajectory optimization. A first experiment was done to show that while direct methods, such as the ones employed in POST, are more robust than indirect methods, they still are not always able to find a solution unless initial guesses are close to the optimal solutions. To illustrate this, Monte Carlo (MC) analysis was done on the initial guesses for the 10 pitch rates. Three Monte Carlo sets were run of 300 cases each. The first MC selected cases within $\pm 0.05°/s$ of the baseline pitch rate value. The second and third

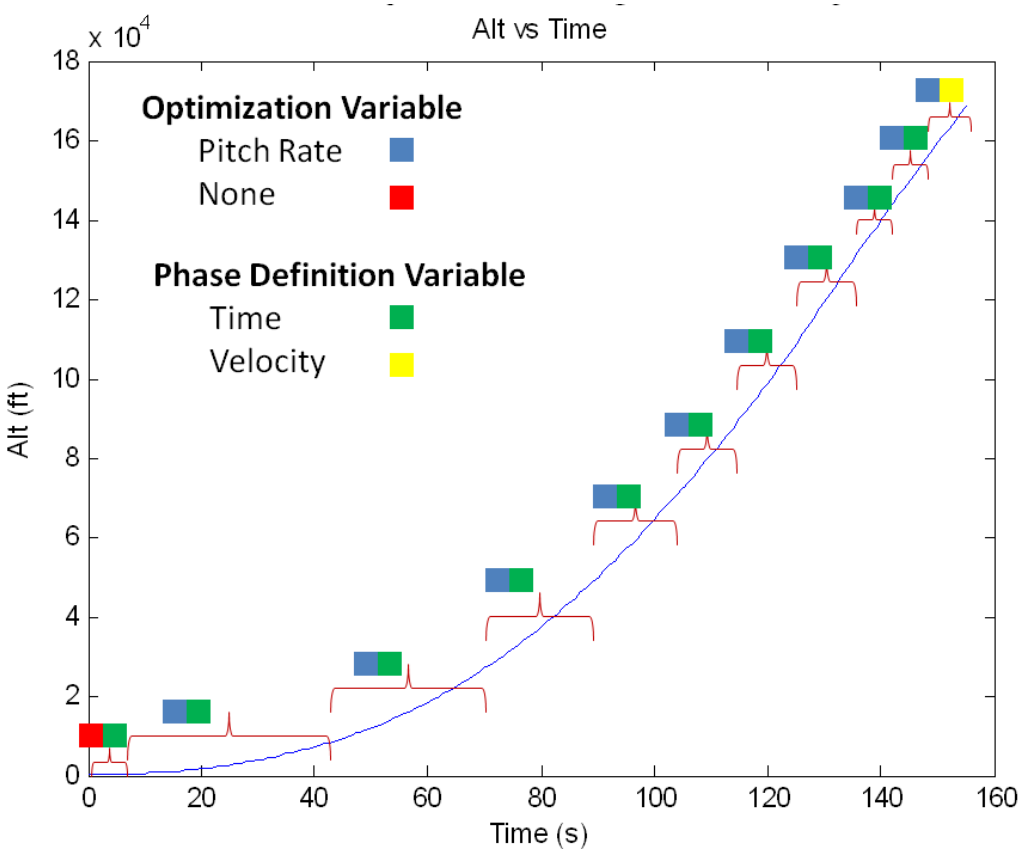
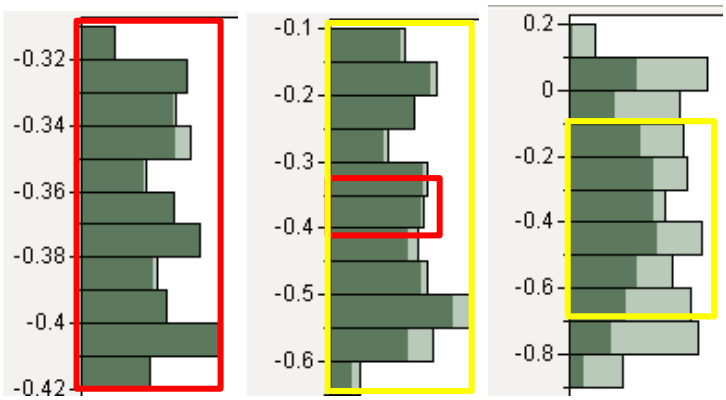


Figure 10: Trajectory discretization graph

had pitch rates ranges of $\pm 0.25°/s$ and $\pm 0.5°/s$ respectively. Figure 11 shows the respective distributions of a sample pitch rate for the MC sets. Table 2 shows the number of successful cases for each MC set. It is interesting to note that from the first to the second MC set the ranges increased by a factor of 5, but from the second to the third only by a factor of 2. From the results, however, a dramatic change in the number of successful runs is seen between the second and third set. As the initial guesses become more varied, direct optimization will have more difficulty finding a solution.

Table 2: Monte Carlo results for case study

MC Set	**1**	**2**	**3**
Cases	300	300	300
Passed	293	274	165
Pass Rate	98%	91%	55%



(a) Set 1 (b) Set 2 (c) Set 3

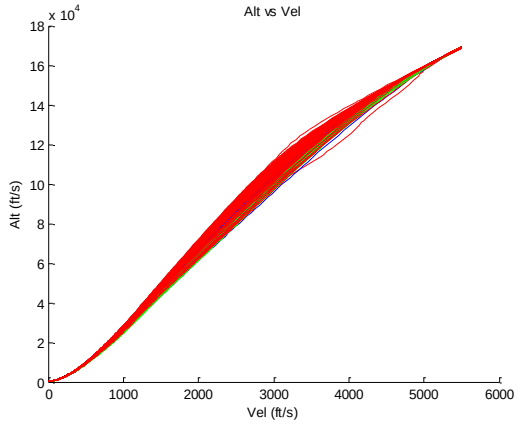
Figure 11: Parameter distribution ranges

One note of interest is that this experiment was run varying only pitch rates. Once phase discretization criteria were included in the optimization process, the percentage of successful run decreased to about 20% for MC set 3.

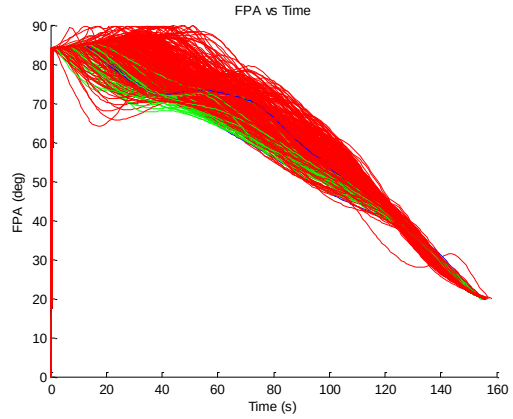
A second experiment was done to show that local optimizers can get stuck in local optima without actually finding the global optimum. A 2000 case MC was run, letting phase event criteria and initial guess values on pitch rates vary. Only about 20% of the cases run found a trajectory that met the constraints. The altitude vs. velocity plot, and flight path angle and velocity vs. time plots are shown in Figure 12. The green trajectories reached the termination criteria with more weight than the baseline, meaning they flew the trajectory more efficiently. Red trajectories were less efficient. The blue trajectories were within 10 *lb* of the baseline weight at the termination criteria.

The main result from these graphs is to show that there are a significant number of not only feasible but "optimum" trajectories. This case study used a relatively simple vehicle and mission, so all the trajectories followed a similar path. However, for more complicated trajectories multiple areas of the design space could include these feasible and "optimum" trajectories.

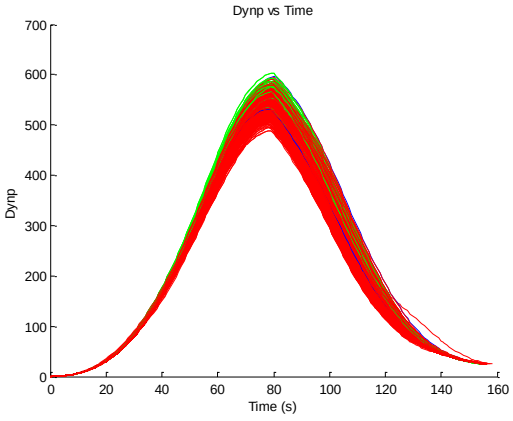
In addition, a significant amount of these trajectories out-performed the baseline



(a)



(b)



(c)

Figure 12: Trajectory plots for MC cases 12(a) Altitude vs. Velocity; 12(b) Flight Path Angle (FPA) vs. Time (note: FPA starts at 85 deg because of oblate spheroid earth model); 12(c) Dynamic Pressure vs. Time

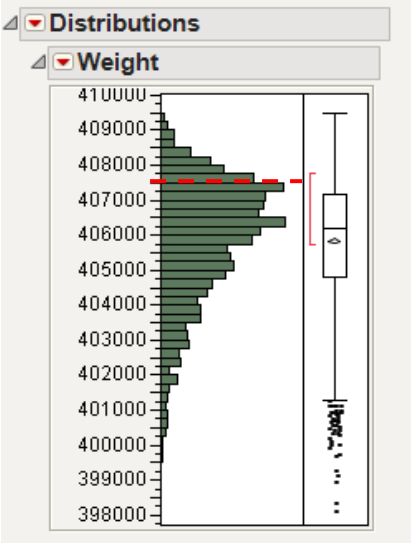


Figure 13: Distribution of weight from 2000-case MC

trajectory. Figure 13 shows the weight distributions. The dashed red line represents the baseline weight. This goes to show that even when a solution is found using a local optimizer, it may not be the global optimum. In fact, there may be a significant amount of better solutions. A global optimizer is important to include in the process, then, to at least increase the probability of finding the best of these local minima.

With the data from the 2000 case MC set, a screening test, or effects screening, was performed. A screening test determines the variability of the response with respect to the variability of the inputs [50]. In other words, which inputs have the greatest effect on a certain output. In a sensitivity analysis multiple order effects are considered. This means that the effect of the initial pitch rate is considered as well as the combined effect of the initial and second pitch rates, for example [41]. The sensitivity study results are displayed in a tornado plot, shown in Figure 14.

As can be seen, the top 5 effects contain mostly phase discretization parameters (i.e. the phase times). This is a very interesting result, because trajectory optimization has historically focused on the vehicle control variables and initial conditions. Here it is shown that discretization variables are important. Any trajectory optimization process should include phase discretization variables in order to fully explore the

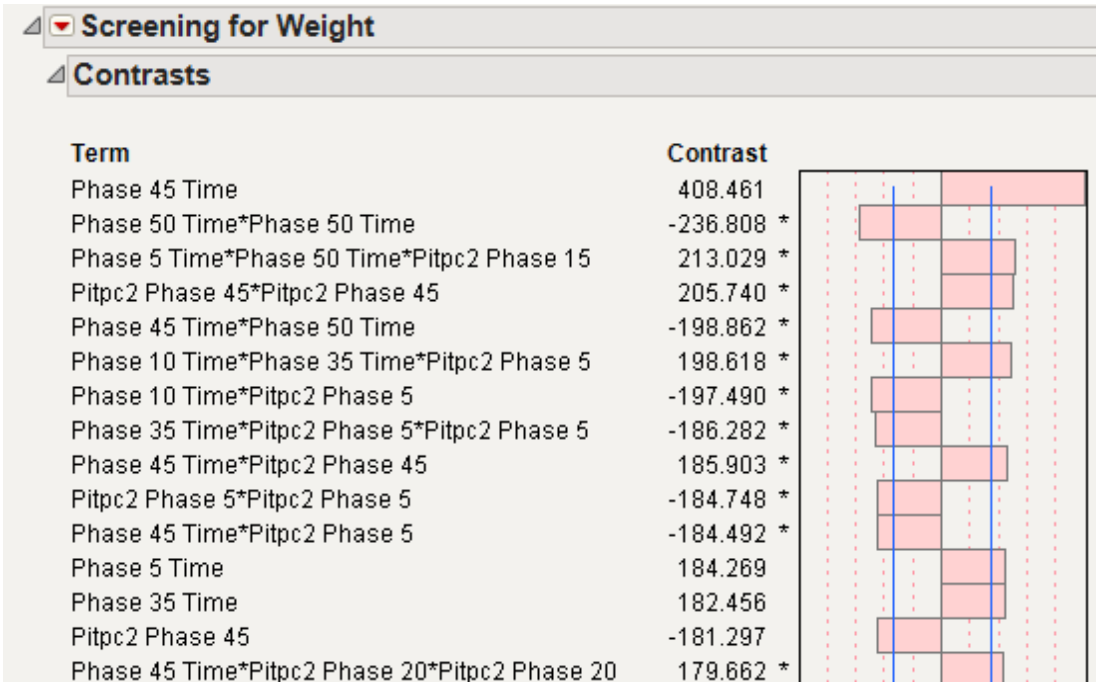


Figure 14: Sensitivity analysis for final vehicle weight

design space.

3.2 Proposed Methodology

In Chapter 2 three observations were made.

1. Using local methods alone for trajectory optimization requires significant user input and does not lead to a robust optimization process.

2. Using global methods alone for trajectory optimization will allow for full design space exploration, but may not lead to the most efficient use of computational resources.

3. Phase discretization variables should be included in the trajectory optimization process.

Based on these observations and the case study, a methodology for trajectory optimization is proposed. Three main aspects of the methodology are as follows:

1. Leverage the design space exploration characteristics of global design methods.

2. Leverage the speed and gradient-based searching capabilities of local methods.

3. Include phase discretization values to explore the entire design space.

Each of these aspects will be discussed in the following sections. Including phase discretization will ensure the entire design space is covered, and combining a global method with a local method will help reduce the human-in-the-loop requirement and increase robustness while still attempting to optimize trajectories in an efficient manner. The hypothesis is that if these three aspects are included in a trajectory optimization methodology, the process will be robust, fast, and not require human-in-the-loop. Recall from Chapter 1 that this is the research objective.

3.2.1 Trajectory Evaluation Approach

Whether a global or local method is used, and whether or not phase discretization parameters are included, a method to evaluate trajectories is required. As with any numerical evaluation, the expense is desired to be as small as possible for the given analysis. The following sections discuss some ways decrease the run-time of the methodology being developed here.

3.2.1.1 Rapid Trajectory Propagator

Because run-time is an issue, it is beneficial to use a very fast light-weight trajectory propagator as the evaluation tool. The trajectory propagators available are POST and OTIS, discussed in Section 2.2.3. These tools involve calculations in several coordinate frames, therefore requiring rotations between the coordinate frames. In addition, path constraints are evaluated at each step, and path constraint values like maximum dynamic pressure and angle of attack are compared to current values at each time step in the process. The number of runs in global methods can be on the order of hundreds of thousands or more, and even saving a small amount of time on

each trajectory propagation could save significant time. The idea of a light-weight propagator is to have a propagator that first does the minimal calculations required to propagate the trajectory and evaluate the ending conditions. Then, trajectories that meet the ending conditions can be reevaluated to determine path constraints. This two-tier filter technique could improve overall run-time for global methods by eliminating a significant number of calculations that are being done on trajectories that will not be used. The improvement will depend on the total number of runs as well as how many trajectories have to be reevaluated. The trajectories can be reevaluated in a global tool to be created, or in a current local tool, as will be discussed later. This leads to the first and second hypotheses.

Hypothesis 1 - In order to perform global optimization on launch vehicle trajectories, a rapid propagator needs to be developed specifically for that purpose.

Hypothesis 2 - Evaluating trajectories without considering trajectory path constraints will significantly speed up the trajectory analysis.

The exact metrics for testing these hypotheses will be discussed in Section 3.3.

3.2.2 Global Optimization Approach

Global methods have the ability to exhaustively explore a design space and provide a robust way of generating feasible trajectories. A good initial guess is not required and global methods do not get stuck in local minima. A robust method that does not require a good initial guess allows trajectory optimization to be performed with

less human-in-the-loop. With local methods humans are required to make the initial guess.

Currently global methods take a significant amount of time because of the required number of cases to be run. This is the cost of design space exploration and robustness. There are several methods, however, that can be employed to try to make the global methods efficient. The goal is to yield information about areas of the design space where trajectories are likely to be feasible and perform well.

3.2.2.1 Sampling Method

When a trajectory design space is considered, there are several ways to initially explore the space. Grid searches and random searches have been discussed previously in this document. However, Latin-hyper cubes and other space filling designs may be more efficient. Determining the right method to select cases for an initial case set is important, as choosing the right sampling method can decrease computational costs. For a given number of cases, more information can be determined from one design over another for a given problem. Studying different design of experiment options will show if there is a significant difference in performance based on sampling method.

3.2.2.2 Design Space Reduction

The trajectory design space can be quite large. There will be areas in the design space that are infeasible, and therefore those areas should not be explored. After an initial set of cases, a reduction of the design space based on the feasible cases can be implemented. The simplest method would be to take the ranges of all the variables for the cases that passed. For example, if each column in a matrix contained the design variables of a feasible case and all the cases that passed are included in the matrix, simply take the maximum and minimum of each row. That becomes the new ranges for the next global search. This method will not necessarily exclude the entire infeasible region, but it will cut down on infeasible space. Using this method, the

initial set of cases must explore the edges of the design space to ensure no areas of feasible design space are excluded.

A second more involved method that can be investigated involves principal component analysis (PCA). PCA is a method that determines linear correlation between data, and finds the minimum number of variables required to express the data [45]. For example, for the data in Equation 11, there exists a pattern where the second column is simply twice the first column. In this case the information in the second column is redundant. The entire set can be obtained using only information from the first column.

$$DATA = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \\ 4 & 8 \end{bmatrix} \tag{11}$$

Any real world problem will be more complex than this, but the principle of the method is the same. Traditionally, PCA is used to reduce the original number of design variables. PCA has been applied in several simulations, including a Vortex Panel Code and Numerical Propulsion System Simulation (NPSS) [45]. Reducing the number of variables, however, is not applicable to the trajectory optimization problem.

Another way PCA could be used by "rotating" the set of design variables to match the shape of the feasible design space. Consider Figure 15. The data points represent the feasible design space, determined after an initial set of cases is run. If the first design space reduction method based only on maximum and minimum values for the design space (x_1 and x_2) were to be used on the data, the dotted green box would be the new design space. There is a significant amount of infeasible design space included (i.e. space where designs are not feasible). However, if the maximum and minimum values of variables v_1 and v_2 are used, the dotted red box results. While

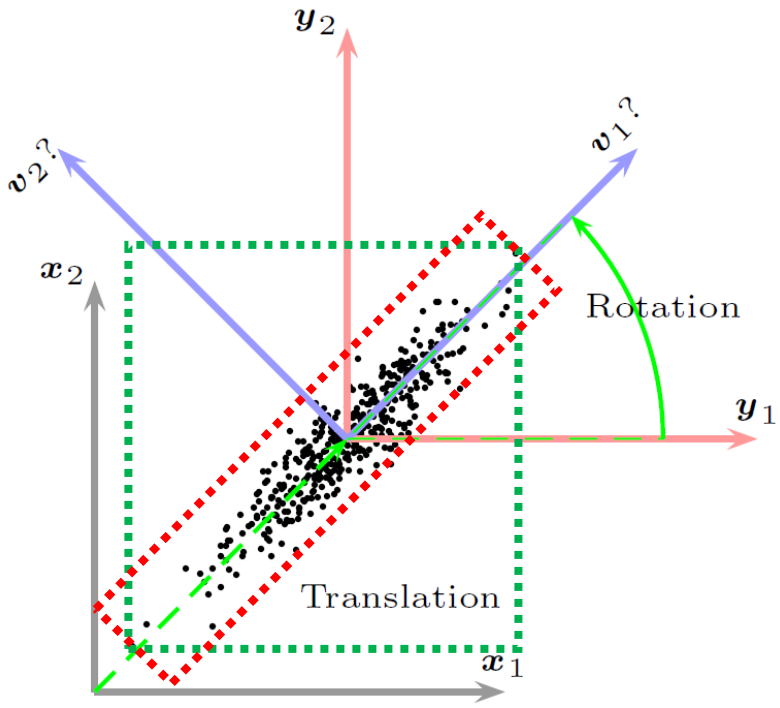


Figure 15: PCA example [45]

there is still some infeasible design space, the total space is much smaller than in the first case, and with a given set of cases, the feasible design space will be explored more thoroughly.

The initial sampling method and design space reduction method both fall under the overall global optimization approach and the hypothesis is given below.

Hypothesis 3 - If an initial sampling of the global design space is performed, the resulting information can be used to create a smaller design space where random cases are more likely to be feasible.

3.2.3 Fine-Tuning Solutions using a Local Optimizer

Local approaches are more efficient at finding locally optimal trajectories than global methods. This means that while the global search is used to explore the space, a local

search should be used to refine the search. Strategically combining global and local approaches will maximize results for a given computational effort [47]. This leads to the next hypothesis.

Hypothesis 4 - If a local search optimization process is applied to the results from the global search, better solutions will be found.

Using local optimization after a global search has been done before in several other fields, including orbit transfer trajectories for spacecraft [8] [63]. There are several options for integrating global and local approaches; two of these options are discussed below.

3.2.3.1 Single Iteration

Perhaps the simplest global/local integration option is to perform a global search, including weeding out those trajectories that do not meet path constraints, and evaluate the resulting best solutions in the local optimizer. This single iteration between global and local methods is in fact straight forward. However, there exists a trade-off between how much the design space is explored before introducing the solutions to the local optimizer. If the search is handed off to the local optimizer too soon there is a risk of missing out on feasible design space because the local optimizer may get stuck in local minima. On the other hand if the global search is performed for too long, computational resources are wasted. A local optimizer may be able to yield the same improvement as a global optimizer in a fraction of the time.

3.2.3.2 Multiple Iteration

Another global/local integration option is to have a feedback between the local and global optimizers. In this case while a random search is being performed, local

searches are refining the best solutions. These can then be handed off to the global method to perform global searches centered on the local optima. It is possible that a local optimizer can significantly improve candidate solutions in much less time than a global optimizer. In a method like this care must be taken to not reevaluate the same design space multiple times. The interplay between global and local can happen continuously as feasible solutions are found or on an iteration-based method. In either case there are many trade-offs between how the methods can be combined.

3.2.4 Phase Discretization

As far as the author can tell, there are no example in literature of trajectory design tools that include phase discretization as part of the optimization process. However, from literature [9] [10] and from the case study, including phase discretization variables in the optimization process is important. As the global and local methods are employed it is important to explore the entire design space. Optimal trajectories from NLP problems are in reality sub-optimal because the trajectory space is broken up into phases [48]. If the manner in which this trajectory is broken up is not explored, a significant amount of the design space is being ignored, without valid justification. Adding phase discretization into the optimization problem does in fact increase the number of design variables, which will lead to longer run-times. However, better results may be found if the problem is run with fewer control variables and more discretization variables. This leads to the final hypothesis.

Hypothesis 5 - If phase structure variables are included in the optimization process, better solutions will be found.

3.2.5 Summary

The trajectory design space will include control variables, such as pitch rates, as well as phase discretization variables. The global search will select a set of cases. These will be evaluated via the rapid trajectory propagator and filtered based on the results. Finally the cases that pass the filter will be optimized via a local method. As discussed in the immediately preceding sections, there are many trade-offs to consider, some of which will be discussed in the following Section 3.3. Figure 16 gives a flow chart representation of the proposed methodology.

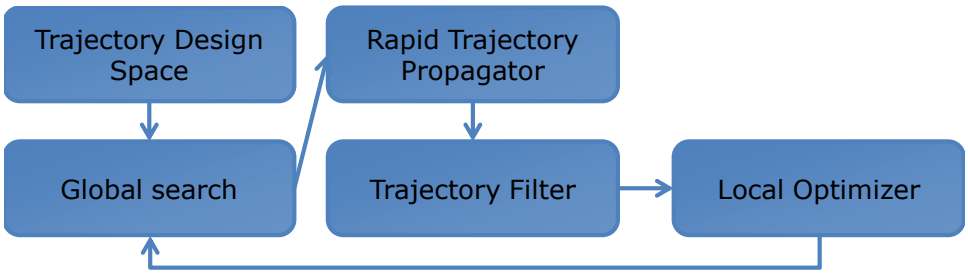


Figure 16: Methodology flow chart

3.3 Experimental Plan

The methodology proposed above raises some questions as to how it can be best implemented. Several research questions are listed below.

1. Can an existing tool be used to evaluate trajectories for a global search?
2. Should path constraints be evaluated during the global search for all the trajectories or at the end of the global search for trajectories that meet the required ending conditions?
3. How can the global search be carried out?
4. Will using local optimization on the results from the global search yield better solutions?

5. Do phase structure variables make a difference in the trajectory optimization outcome?

3.3.1 Experiment 1

The first experiment is proposed to answer the question of what to use as the rapid trajectory propagator (Research Question 1). The rapid propagator is used to determine where trajectories end given initial conditions and controls. There are essentially two options: use a legacy code in propagator mode only or develop a new trajectory propagator. Current tools would be used by essentially running the tool without the optimization process (i.e. as a simple propagator). While this will cut down on the development time, this option may not be the best because the current tools were not designed as pure propagators. It is expected that a tool designed and developed as a simple propagator will perform much faster, and thereby lead to a faster overall trajectory optimization process. In order for global searches to be feasible, it is expected that the tool should be able to run on the order of $100,000$ cases overnight.

3.3.2 Experiment 2

The second experiment will determine when trajectories should be filtered (Research Question 2). This experiment will compare evaluating the constraints during the global search vs. at the end of the global search. The comparison will lead to determining which method is appropriate given the total number of cases and the expected number of cases that meet the final conditions. (If all cases are expected to pass, there is no benefit to running the cases a first time to determine ending conditions and then running them again to evaluate path constraints.) Like Experiment 1, the results of this experiment will determine the best option to speed up the overall optimization process.

3.3.3 Experiment 3

The third experiment will consider how to initialize the global search and reduce the design space as needed (Research Question 3). There are two parts to this experiment. The first part is investigating how to initialize the global search. As discussed previously in this study, there are many options, including grid search, random search, Latin-hypercube, etc. These can be compared and evaluated based on run-time and quality of results. In light of scoping this problem, only two types of designs will be compared. This will be discussed in detail in Section 4.4.

The second part deals with reduction of the design space during the global search. It may result that the design space reduction strategies in fact afford no benefit and that a larger initial global search is a better option. This could be for one of two reasons. The first would be if the design space is too large given the initial set of cases. In this case the feasible design space will not be well defined, and any reduction in the total design space may be excluding significant amounts of feasible design space. The second reason would be if the design space was small enough to where most of the design space is feasible. In this case a smaller initial case set would be recommended. Comparing which, if any, design space reduction method is more effective will afford the overall trajectory optimization process speed and better results.

3.3.4 Experiment 4

The fourth experiment will answer the question of how to integrate the local and global optimizer (Research Question 4). A comparison will be made between an integrated local method (i.e. feedback between local and global method) and a simple global to local single iteration approach. The goal of the local optimizer is to find areas of the feasible space that contain the best performing trajectories. Figure 17 shows a flow diagram of a single iteration integration scheme vs. a feedback or multiple iteration integration scheme.

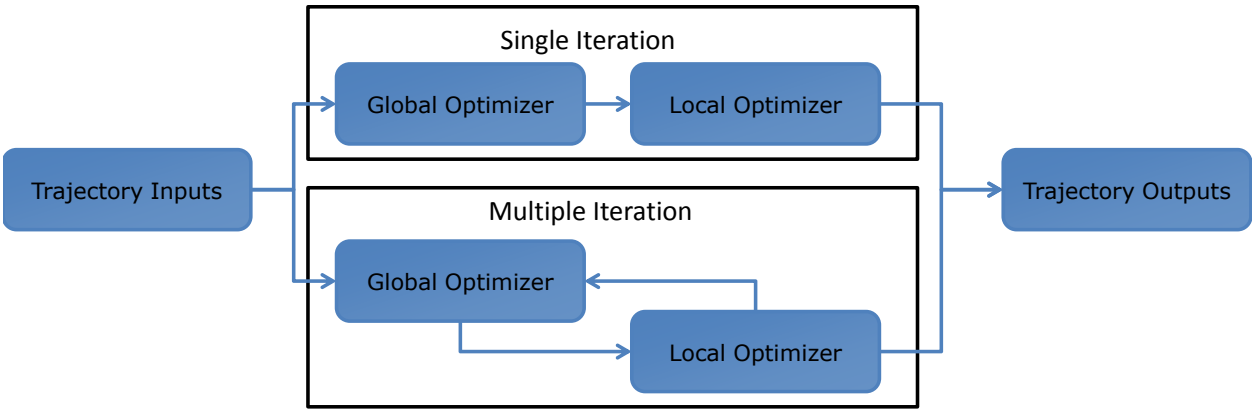


Figure 17: Integration of global and local optimization

3.3.5 Experiment 5

The fifth and final experiment answers the question of whether or not phase discretization affects the outcome of the optimization or search process (Research Question 5). The experiment will be to essentially re-run some of the cases sets from Experiment 4 while changing some of the phase discretization variables, such as number of phases and the times those phases occur, and see if the results are better or worse. It should be noted that while benefits may be seen, the cost of those benefits must be considered. If adding a phase doubles the time it takes the local optimizer to run, for example, it may not be worth the improved performance.

3.4 Methodology and Experiments Summary

This chapter validated the observations from Chapter 2 using the case study in Section 3.1. Based on the observations and the case study, research questions and hypotheses were developed, and an experimental plan to test the methodology was implemented. Chapter 4 will show and discuss the results of the experiments.

CHAPTER IV

RESULTS

In this chapter, a sample problem is developed on which to perform the experiments and test the methodology. The reason the problem from Section 3.1 was not used again was because it did not go to orbit, only to a staging point. In addition, the problem in the case study was for Reusable Booster system. The problem in Section 4.1 was developed as a more representative problem for generic launch systems (i.e. not reusable). Also in this chapter, the experiments are performed and discussed.

4.1 Example Problem Trajectory Formulation

Evaluating the methodology and generating results requires a sample vehicle and mission. The objective of this thesis is to show that the methodology developed can show improved results in terms of finding feasible trajectories for new problems without requiring significant human-in-the-loop. The sample vehicle and mission, then, must be complex enough to capture the challenges of trajectory optimization. However, choosing a trajectory that is too complex will complicate the analysis process without contributing to the objective of this thesis.

With that in mind, the Delta IV Medium vehicle was chosen as an example problem for the experiments. The mission was selected directly from the LEO Capability plot for the Delta IV Medium [39]. It is important to note that for the purposes of this discussion, no information about the trajectory was input into the problem. The methodology is designed to optimize the trajectory for a new vehicle; i.e. no information about the trajectory is known.

The Delta IV Medium was developed by Boeing as an upgrade of the Delta II. The first stage of the Delta IV Medium, known as the Common Booster Core (CBC), is a

liquid oxygen (LOX) hydrogen stage powered by a single RS-68 Rocketdyne engine. The Delta IV family was designed in a modular fashion around the CBC to service different missions and payloads. This is achieved by strapping on up to 4 solid rocket motors for variations on the Delta IV Medium or by attaching 3 CBC's together for the Delta IV Heavy. The second stage is different for the Delta IV Medium and its variants and the Delta IV Heavy. In both cases, the upper stage is a LOX hydrogen stage powered by a single Pratt & Whitney RL10B-2 engine [39].

The Delta IV Medium was selected because it was the simplest of these vehicles, while still representing a typical earth to orbit vehicle. The mission selected is to a $400\,km$ circular orbit. The CBC burns until no propellant remains, throttling only to control max acceleration, and the upper stage lights and burns until orbital velocity is reached. While this may not be the best way to achieve this orbit (a two burn upper stage trajectory may be better), it serves its purpose as a test bed for the trajectory optimization methodology.

Following is a breakdown of the vehicle and trajectory definition used in this sample problem. It is important to remember that testing this methodology does not require that the model used perfectly reflects the Delta IV Medium. What is required is a vehicle and mission that capture the challenges associated with trajectory optimization, namely the large non-linear design space. The values used to model the Delta IV Medium and its sample mission were gathered from publicly available data. Table 3 gives the weight breakdown used to model the vehicle. The vehicle propulsion parameters are given in Table 4.

Aerodynamic data for the Delta IV Medium is not publicly available. However, some form of aerodynamic data is necessary to simulate any earth to orbit trajectory. The Air Force Research Laboratory (AFRL) has developed a semi-empirical design tool (Missile DATCOM) to calculate aerodynamic data for a wide variety of different vehicle configurations and flight conditions [69]. MDATCOM is intended as

Table 3: Delta IV Medium vehicle weights [39]

Vehicle Element	Weight (kg)	Weight (lb)
Core Burnout	26760	58996
Core Propellant	181074	440042
Upper Stage Burnout	2850	6283
Upper Stage Propellant	20400	44974
Payload Fairing	6300	3697
Payload	8500	18800

Table 4: Delta IV Medium vehicle propulsion parameters [39]

Propulsion Element	First Stage	Second Stage
Engine	RS-68	RL10B-2
Vacuum Thrust (lb)	751000	24750
Vacuum ISP (s)	409	462.4
Exit Area (ft^2)	49.9	48.9

a preliminary design tool for missiles [17]. However, many launch vehicles, including the Delta IV Medium, are similar in shape to missiles. Using non-dimensional aerodynamic coefficients allows scaling between smaller missile size and larger launch vehicles. MDATCOM has been used for launch vehicle aerodynamic calculations [64], and can be used to estimate the aerodynamic data necessary to model the Delta IV Medium.

MDATCOM takes as input a set of points that represent distance along an axis and the corresponding distance from that axis. The vehicle profile is represented by this set of points, and the profile is rotated around the axis to generate an axisymmetric representation of the launch vehicle. The Delta IV Medium was modeled using Vehicle Sketch Pad (VSP), a NASA open source parametric geometry tool. Figure 18 shows the layout.

Figure 18: Delta IV Medium model

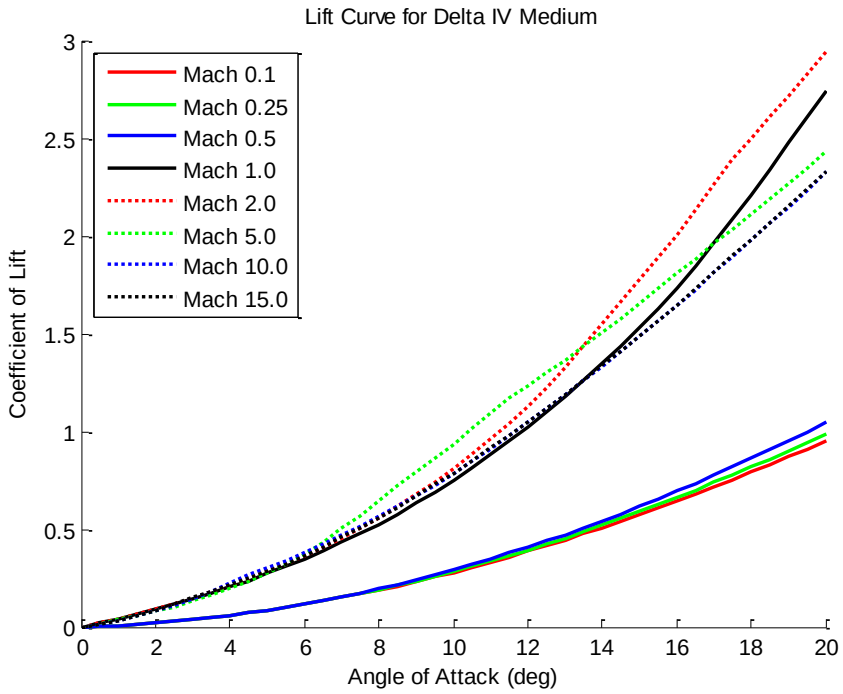


Figure 19: Delta IV Medium lift coefficients at varying flight conditions

MDATCOM outputs coefficients of lift and drag for each of the input flight conditions. The flight conditions of interest for this problem are angle of attacks between $-20°$ and $20°$ and Mach numbers between 0 and 15. The vehicle will experience Mach numbers higher than this during its trajectory, but only when it has reached an altitude where aerodynamic forces become negligible (less than 1 lb). Figure 19 and Figure 20 show the aerodynamic data for the Delta IV Medium model at relevant flight conditions.

After defining the vehicle using component weights, propulsion parameters and aerodynamic data, the sequence of events for the mission is input. Figure 21 depicts the sequence of events for the selected mission.

The trajectory is controlled using inertial pitch rates. The vehicle launches from Florida's east coast. After vertical rise of 3500 ft to clear the launch structure, the first pitch rate is used to initialize the gravity turn. After the gravity turn is started, the vehicle flies at $0°$ angle of attack and sideslip angle. The second pitch rate occurs

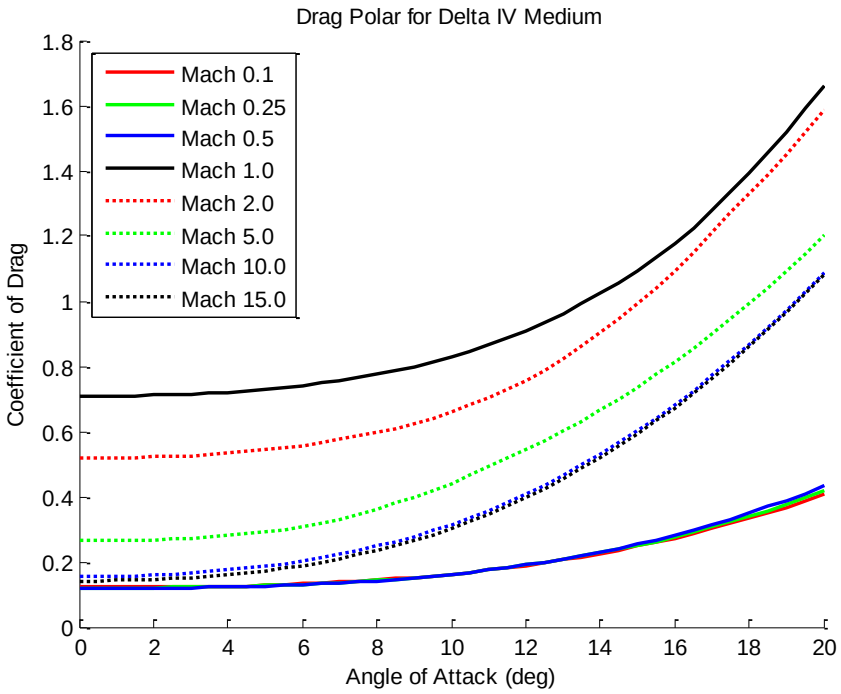


Figure 20: Delta IV Medium drag coefficients at varying flight conditions

after the dynamic pressure has reached its peak and been reduced to $20\,lb/ft^2$. These two pitch rates control the vehicle until the first stage burns the available propellant and is jettisoned. Two seconds after the first stage is jettisoned the upper stage ignites and has 3 pitch rates available to control its flight until it reaches orbital velocity. The payload fairing is jettisoned 10 seconds after the first stage is jettisoned. Normally the payload fairing is jettisoned when the free molecular heating rate, given in Equation 12, is equal to $0.1\,Btu/ft^2s$ and decreasing [4]. For this vehicle and mission, though, that point occurs before the first stage is jettisoned. Thus, the payload fairing jettison was modeled based on time.

$$FMHR = c \times q \times v_{rel}$$
$$\text{where } c = 0.00128593 \tag{12}$$

The phases and their specific values of the trajectory are given in Table 5, where Time is measured in seconds, Alt (altitude) in feet, q (dynamic pressure) in psf, $wprp_1$ (first stage propellant) in lb, and Velocity (inertial) in ft/s. An important note

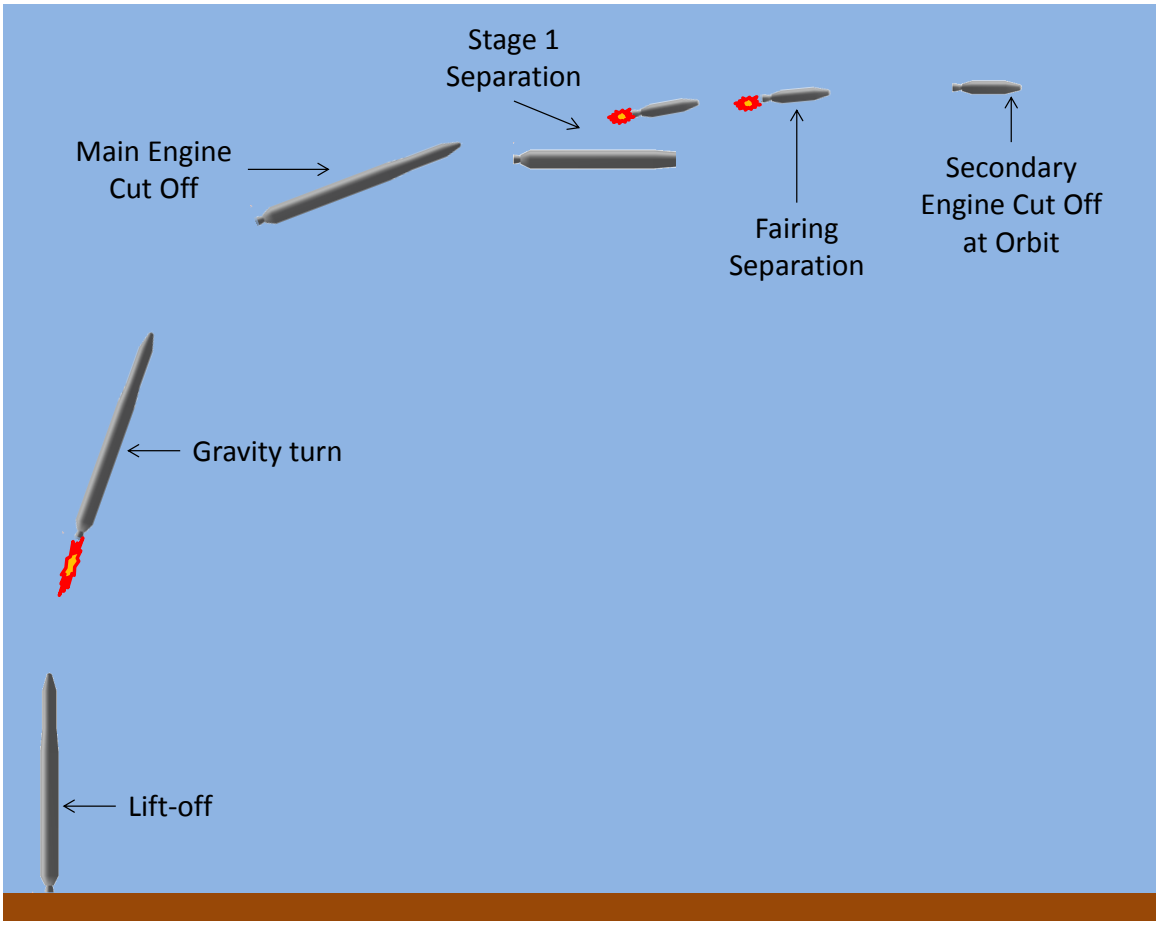


Figure 21: Sequence of events for Delta IV Medium sample mission

regarding the "Control" column is that control here is used to refer to a parameter that the optimization process is in control of. In Phase 4, for example, there is guidance occurring to keep the angle of attack at zero, but the optimizer has no control over this parameter, and therefore it is not a control. For this study, the parameters of concern are the controls, not the guidance variables.

Table 5: Phase structure for trajectory of sample problem

Phase	Description	Start		End		Control
		Criteria	Value	Criteria	Value	
1	Initialization and vertical rise	Time	0	Alt	3500	None
2	Gravity turn	Alt	3500	q	150	$u1$
3	Reduce alpha	q	150	Time[i]	10	None
4	Max dynamic pressure	Time[i]	10	q	20	None
5	First stage guidance	q	20	$prop_1$	0	$u2$
6	First stage burnout	$prop_1$	0	Time[ii]	2	None
7	Upper stage guidance	Time[ii]	2	Time[iii]	200	$u3$
8	Upper stage guidance	Time[iii]	200	Time[iii]	500	$u4$
9	Upper stage guidance	Time[iii]	500	Velocity	25548.8[iv]	$u5$

This concludes the formulation of the vehicle and mission model. Again, it is important to remember that the purpose of this example problem is not to perfectly reflect the Delta IV Medium's performance or characteristics. Instead this example will be used as a baseline model to test the various elements of the trajectory search and optimization methodology being investigated in this thesis.

[i] Time here is measured from the beginning of the phase
[ii] Time here is measured from first stage burnout
[iii] Time here is measured from 2 seconds after first stage burnout
[iv] The velocity required for circular orbit

4.2 Experiment 1: Evaluation of Trajectories

Once a vehicle and mission have been formulated the next step is to evaluate the trajectory. As discussed in Section 2.2.4 there are many options. Recall that evaluating a trajectory and optimizing a trajectory are very different processes. The evaluation of a trajectory requires only a numerical integration method, whereas optimizing a trajectory requires some optimization algorithm in addition to the numerical integration method. Research Question 1, repeated below for clarity, is answered in this section.

Research Question 1 - Can an existing tool be used to evaluate trajectories for a global search?

The only existing tool available to the author for this thesis was POST, discussed in Section 2.2.3.1. Originally the author hypothesized that creating a light-weight Rapid Trajectory Propagator would drastically outperform a tool like POST. POST requires the writing and reading of large input and output files every run, and because the author does not have access to the source code there was no way to tell how much overhead POST requires in the way of setting up each problem for trajectory evaluation. POST was designed as an optimization tool, not a mere trajectory propagator [19]. In light of all that, it was expected that the author could construct a propagation tool that would perform better.

However, based on the research objective of this thesis, and upon recommendation of the committee, the decision was made to determine if POST was a viable option for this study, and if so to use it without creating a trajectory propagator from scratch and comparing it to POST. There were two main reasons for this. The first is that the research objective can be achieved without a new propagation tool. The goal

is to determine a methodology for feasible trajectories to be found and improved upon without human-in-the-loop and in a timely fashion. Once a methodology that achieves this is created, any software advances can be plugged in to the different parts of that methodology to increase its speed or efficiency. However, the methodology can and should be created by comparing how given methods in the methodology perform. The second reason is that the creation of a tool would required a significant investment for the software development, debugging, and validation. All this while there is no guarantee that it will outperform the current tool. Keeping the scope of the problem in mind, it was determined that if POST is a viable option, which will be discussed in the following paragraphs, it should be used.

Whatever tool is used to evaluate trajectories, the experiments require a large number of cases to be run, which requires the tool to be automated. This automation must be able to input desired cases into the tool, run the tool, and gather any outputs that are required. Originally POST was automated using MATLAB. Each case took about 2 s to run. Much of this was the overhead required by MATLAB to write the input files for POST. The author decided to implement the POST automation in the Python language to speed up the run-time. Python was chosen because of familiarity with the language. Another language may be more efficient for this automation, but that study is outside of the scope of this thesis. Running the cases in Python lead to a six-fold increase in run-time, with each case taking about .33 s. Obviously more complicated POST trajectories will require a longer run-time per case. For this study, a run-time of .33 s is sufficient. Three other metrics of a process automation that were considered were repeatability, consistency, and scalability.

The test for repeatability will be discussed first. This test involved running the same set of cases multiple times and comparing the outputs. As expected, the results were exactly the same.

Table 6: Run-times for consistency test

Case Set	Number of Cases	Total Run-time (s)	Average Case Run-time (s)
1	1000	137.91	0.14
2	1000	136.59	0.14
3	1000	137.11	0.14
4	1000	137.28	0.14
5	1000	137.94	0.14
6	1000	140.57	0.14
7	1000	139.09	0.14
8	1000	137.98	0.14
9	1000	137.37	0.14
10	1000	137.84	0.14
Average	1000	137.97	0.14

The test for consistency is essentially to see if different sets of the same number of cases are run, how consistent the overall time required for the runs is. Each case is for the same vehicle and same mission, but with a different initial guess on the control vector. Evaluating the consistency of the run-time will help with planning out experiments and determining how many cases can be run. It is not necessary, but it is expected and desirable. In order to test consistency, 10 different sets of 1000 cases were evaluated, and the run-time per case for each of the 10 sets was compared. Table 27 shows the results of testing for consistency and it can be seen that the run-time is consistent.

Finally, scalability is a discussed. It is necessary that the automation process and run-time be scalable to allow for a large number of cases to be run, possibly over a weekend or an entire week. At .33 s it is possible to run over 1.8 million cases in a week. If the code slows down as it runs more cases, it can lead to very long run-times for the final cases of large case sets. To test scalability, 5 different sets of cases were run. The sets had 10, 100, 1000, 10000, and 100000 cases respectively. It is expected that each set will take about 10 times longer than the previous set. Some small differences are expected, regarding how long it takes to read in the cases to be run and how long it takes to compile all the results after running all the cases. Based

Table 7: Run-times for scalability test

Number of Cases	Total Run-time (s)	Average Case Run-time (s)	Run-time Ratio
10	2.76	0.17	–
100	15.37	0.14	5.6
1000	138.09	0.14	9.0
10000	1382.17	0.14	10.0
100000	14167.02	0.14	10.2

on the results shown in Table 7 it can be assumed that running a set of 10000 cases for example will be representative of any set of 10000 cases for that specific vehicle and mission.

Table 7 shows the results of the scalability experiment. As mentioned earlier, there is some overhead required each time a set of cases is run. This overhead included setting up any necessary folders and files as well as making sure any old files are removed. As more cases are run this overhead required will be a smaller percentage of the overall run-time. For the sets with a higher number of cases it can be seen, as expected, that the run-time ratio approaches 10, which is the ratio of the number of cases. As the number of cases increases, the run-time increases linearly with number of cases. This trend can be seen in Figure 22, plotted using log-log scale for easier visualization.

Based on the results of these experiments, POST is indeed a valid option for evaluating these trajectories. The answer to Research Question 1 is yes. Hence, there is no need to create another trajectory propagator, and the other experiments can be conducted using POST.

After evaluating POST as an option for the evaluation of trajectories, the automation code and the POST input files themselves were modified to increase the run-time (for example, comments were removed from the POST input files that got written every run). For this reason, the run-times used in later experiments are slightly better than the run-times here.

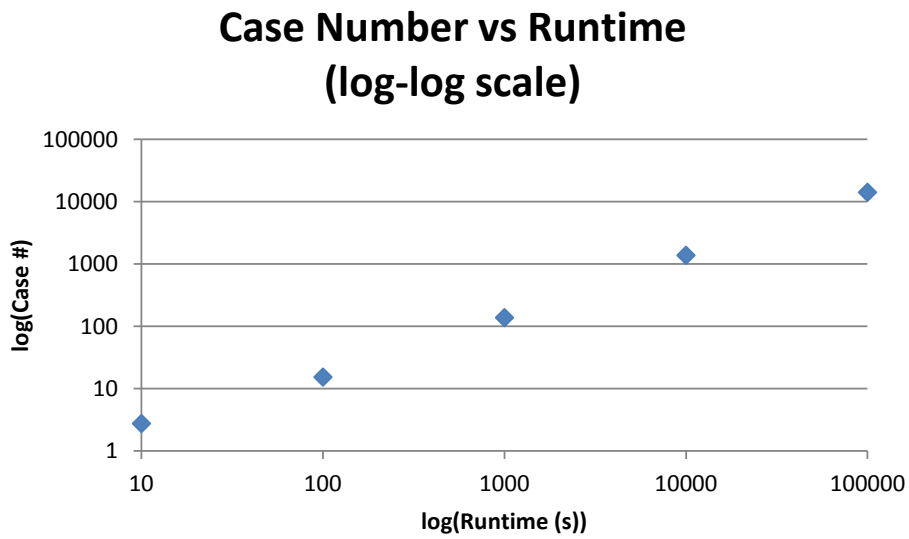


Figure 22: Run-time vs. number of cases from scalability test

4.3 Experiment 2: Filtering Trajectories

Once the method for evaluating trajectories has been determined, testing the different options for the methodology can begin. Research Question 2, repeated below, addresses how the trajectory evaluation tool should be used to find feasible cases, or feasible sections of the design space. Evaluating path constraints at each time step during the trajectory evaluation may be expensive, and it could be of benefit to only evaluate path constraints for cases that actually meet the ending constraints.

Research Question 2 - Should path constraints be evaluated during the global search for all the trajectories or at the end of the global search for trajectories that meet the required ending conditions?

This question is essentially asking if it is more computationally efficient to evaluate path constraints for all the cases run, knowing that in the global search there will be

many infeasible cases, or should only the trajectories that meet the ending criteria be re-evaluated to determine path constraint values.

Experiment 2 was set up to test how path constraints should be evaluated. A set of $100,000$ cases was created. The set of cases was run the first time evaluating constraints for each trajectory, and the second time without evaluating constraints. Because of run-time differences due to the computer condition, the whole experiment set up was executed twice for a total of 4 set runs, 2 evaluating path constraints and 2 without evaluating path constraints.

It is worth noting this Research Question was tied to Hypothesis 1, which predicted that an in-house developed trajectory propagator would perform faster. The source code would obviously be available for this tool, and modifications could be made to optimize how the trajectories are propagated. It was in this context that a faster trajectory evaluation was expected by not including path constraints. However, because a legacy tool was used to evaluate the trajectories (see Section 4.2), the author does not know how path constraints are handled from a programming perspective, and it is assumed that the legacy code is not optimized to run without evaluating path constraints. Accordingly, a major difference in run-time between trajectories with path constraints and trajectories without path constraints is not expected.

Table 8 shows the results of Experiment 2. These results reflect run-times based on trajectories that are not optimized. It is expected that comparing the optimization of trajectories with and without path constraints would show a larger run-time difference. However, this endeavor would be for academic purposes, because an optimized trajectory without relevant path constraints would be of no value.

There is not a major difference in run-times between evaluating the trajectories with or without constraints. The difference is on the order of $5\,min$. At $0.11\,sec$ per case, however, over $2,500$ cases can be evaluated in $5\,min$. This becomes a question, then, of how many cases are expected to pass. Essentially, if 2.5% or fewer cases

Table 8: Path constraint evaluation comparisons for non-optimized trajectories

	Without Path Constraints		With Path Constraints	
Repetition	Total Run-time (s)	Average Case Run-time (s)	Total Run-time (s)	Average Case Run-time (s)
1	10684	0.11	11132	0.11
2	11034	0.11	11101	0.11

are expected to pass, it becomes worth it to evaluate trajectories without constraints first, and then only evaluate the ones that meet the ending criteria with constraints.

The question of how much benefit is realized by doing this is of importance. If $100,000$ are being run, as in the case, a few minutes can be saved. However, if $10,000,000$ cases are run, this approach could save hours. It may be beneficial to run a small number of cases, say 1000, to get an approximate percentage of how many cases will pass. This will allow the user to make an educated guess on the pass rate and set up the run accordingly.

As a follow-on to this experiment, it was observed that if the same set of cases was run in the optimization mode (i.e. optimizing each case), only the cases that reached the trajectory ending criteria, in this case inertial velocity (see Section 4.1), were able to be optimized. The way POST works requires an initial trajectory that reaches the ending condition and passes through all the defined phases before it can optimize it. This observation will be important in future discussions of how to combine global and local approaches.

The conclusion to Research Question 2 is that the path constraints should be evaluated in the global search. The amount of time it would take to go back and rerun the cases that did pass is on the order of the time saved, and therefore the extra effort is not worth it. A future tool, however, designed to run without path constraints may be able to significantly decrease run-time for cases run without path constraints.

4.4 Experiment 3: Global Search Methodology Formulation

The first part of the proposed methodology is a global search of the relevant design space. As a reminder, the relevant design space is composed of trajectory control variables, such as pitch angles or rates. Phase discretization variables are not included until Experiment 5. Vehicle variables, such as engine thrust or propellant mass, are not considered here.

There are a number of ways to initialize a global search. Research Question 3 is the question addressed in this section. Based on the discussion in Section 3.3.3, Research Question 3 has been split into two subquestions to address two different aspects of the global search.

Research Question 3a - How can the global search be initialized?
Research Question 3b - What is an effective strategy to reduce the design space?

Research Question 3a addresses the first step in the global search. The goal with a global search is to consider the entire design space. While this is not possible with a continuous design space, there are methods that are designed to cover more of the relevant design space. The discussion in section 4.4.2 will help clarify this point.

Research Question 3b relates to why a global search is being run. The goal is to pass information on to a local search. Ideally, the design space will be reduced so the local search can be performed over a much smaller design space. From this perspective, it is desired that a high number of cases from the global search be candidates for a local search so the feasible space can be defined.

4.4.1 Local Search Candidate Trajectories

A set of criteria must be developed to determine if a case from the global search is a candidate for the local search. The way POST works, only the cases that meet the trajectory ending condition, based on inertial velocity for this problem, will have a chance of being optimized by POST (recall this was observed in Section 4.3). This leads to a very simple way of determining whether or not a case has passed: did it reach the ending condition. The cases that reach the ending condition in POST and have a possibility of being optimized by POST will be referred to as feasible cases.

A small experiment was performed to confirm this. A set of $100,000$ cases was run without using the optimization tool in POST. The trajectories were initialized and propagated. Of these $100,000$ cases, only 134 reached the ending condition. Originally, it was expected that the final altitude and flight path angle, the two other conditions used to define the final orbit, would have to be within a certain tolerance of the target values for POST to be able to optimize the trajectory. However, after running the same $100,000$ cases using the optimization tool in POST, only the 134 cases that reached the ending condition were able to be optimized by POST.

Not all of these 134 cases, however, were able to reach the target orbit. It may be of interest to determine if there are any limitations on the altitude or flight path angle reached during the non-optimization propagation for cases that reached the target orbit when run with the optimization option.

Figure 23 shows the distribution of gdalt (altitude) and gammai (inertial flight path angle) for the 134 cases that reached the ending condition when run without the optimization option. If limitations exist on the altitude or flight path angle for a case to reach the target orbit when run in POST with the optimization option, a distribution with a smaller range would be expected for the cases that passed.

Figure 24 shows the same distributions of the original ending altitude and flight path angle values for cases that reached the ending condition and also reached the

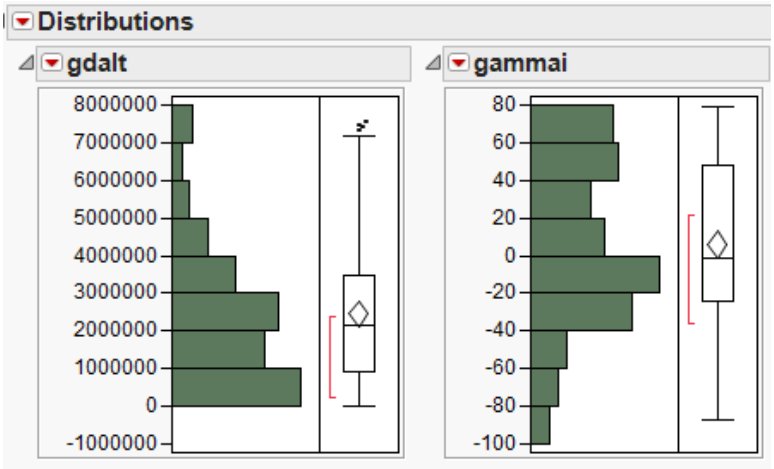


Figure 23: Cases that reached the ending condition when run in POST without optimization

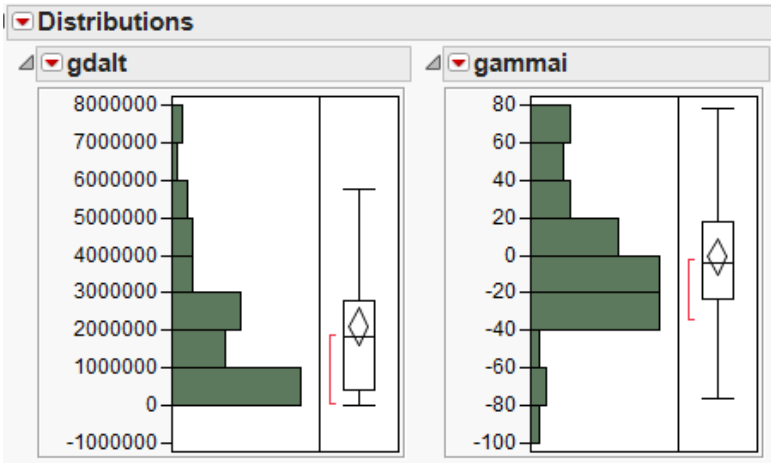


Figure 24: Cases that reached the ending condition when run in POST without optimization and reached the target orbit when run in POST with optimization

target orbit when run in POST with optimization. The ranges on the distribution are for all practical purposes the same, and by inspection the distributions are of similar shape. This implies that there is not a particular range of altitudes or flight path angles that would make it more likely for a case to reach the target orbit given it reaches the ending condition. This can be confirmed in Figure 25 by looking at the distribution of the cases that did not reach the target orbit after being optimized, but did reach the ending criteria in the global search. Again, the ranges are the same and the distributions are of similar shape.

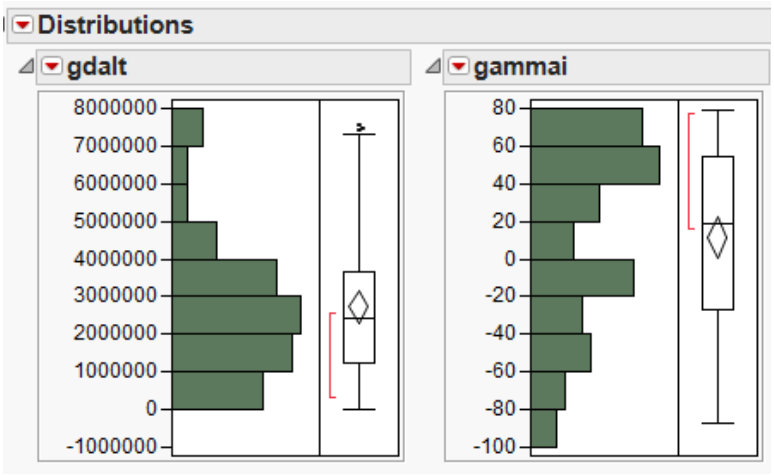


Figure 25: Cases that reached the ending condition when run in POST without optimization and did not reached the target orbit when run in POST with optimization

From this it can be concluded that any case that reaches the ending condition when run without optimization is possibly a case that will reach the target orbit when run with optimization. In terms of global and local searches, any case that reaches the ending condition in the global search should be evaluated using a local method (the optimization option in POST). It can also be concluded that there is no hard limit on the constraint values, in this case altitude and flight path angle, that will determine whether or not the cases that reached the ending conditions without optimization will reach the target orbit when run with optimization.

4.4.2 Design Space Coverage

When conducting a series of experiments, a decision has to be made as to which experiments to run. A field of study known as Design of Experiments has been researched and developed by statisticians. The goal of Design of Experiments is to maximize the amount of information obtained about the system in question for a given number of experiments [27]. For example, running Set A of 10 experiments may yield much more information than running Set B, even though it may also have 10 experiments. There are many other advantages to be gained from Design of Experiments. For a more complete discussion on the subject refer to Dieter [27].

There are several options to choose from when designing an experiment. Following is a discussion of several of these options. This discussion is based of the lecture notes from the Advanced Design Methods class taught by Dr. Mavris at Georgia Institute of Technology in the Fall of 2010 [50]. In general, there are two types of experiments. One type is a structured type, where the design will look like a structured pattern. The other type is random, where points are chosen randomly, but with some overall goal in mind. The following discussion will clarify this distinction.

Possibly the most intuitive design is known as a full-factorial design. This design is a structured design and involves discretizing the continuous variables and running every possible combination at each variable level. For example, if variables $A1$ and $A2$ are discretized into two levels each, a full-factorial design would involve 4 experiments: $A1$ high with $A2$ high and low and $A1$ low with $A2$ high and low. The number of experiments in a full factorial is given by Equation 13

$$\#\ of\ Experiments = \#\ of\ Levels^{\#\ of\ Variables} \tag{13}$$

This design does cover the design space, but with a lot of variables the number of experiments required can become infeasible. For the example problem described in Section 4.1 there are 5 variables. Two levels would not be enough to adequately map the design space. If 20 levels are used, the full factorial design would require $3.2\,million$ experiments. One hundred levels would require $10\,billion$ experiments. If 20 levels are used with 6 variables, $64 million$ experiments are required. While the full factorial design is good at covering the entire design space, it suffers from the "curse of dimensionality", where the number of experiments required becomes infeasible as the number of variables and levels increases. It is worth noting there exist fractional factorial designs, where not all the possible combinations are run. These, however, do not cover all the corners of the design space, and are not discussed any further in this document.

Central composite designs are another type of structured design that combine a 2

level full factorial with a series of points at the midpoint of each variable range as well as a series of points with each variable but one at the midpoint of each variable range and the excluded variable set to its high or low value. Central composite designs are good at covering the corners of the design space, but may leave large areas of the interior empty.

There is a set of experimental designs known as space-filling designs. These designs are set up randomly. The first of these is known as sphere packing. The experiments are designed so as to be as far away from any other experiment as possible. This leads to a good coverage of the interior of the design space, but a lack of experiments in the corners.

Uniform designs are another type of space-filling designs. The goal of a uniform design is to have equal separation between all the points. Again, there is good coverage in the interior of the design space, but there can exist regions of the design space that have poor coverage.

The last type of space-filling designs discussed here is the Latin hypercube (LHC). This design splits each variable into a number of bins equal to the number of experiments required. Then it guarantees that each bin of each variable will have a point in it. This method covers the interior of the space very well, but may not have points in the corner of the design space.

It is worth mentioning the concept of orthogonality at this point. An orthogonal set of experiments is one where the variables are linearly independent. With linearly independent variables there is no linear correlation between the variables. This is important in the design of experiments to ensure that any dependence seen in the outputs comes from the behavior of the system and not from what set of inputs were run. Fractional factorial and central composite designs are inherently orthogonal. Space-filling designs are not, but can be set up in such a way as to minimize the linear dependence between the variables.

4.4.3 Global Search Initialization

4.4.3.1 Initialization Options

Based on the discussion above, and considering the scope of this study, a comparison was done between 2 different ways to initialize the global search: full factorial and Latin hypercube. These two designs were chosen as representatives of the two main types of designs discussed. The full factorial was set up using 20 levels, resulting in 3.2 *million* experiments. The Latin hypercube was run with 3.2 *million* experiments as well.

The comparison of these two methods will be conducted based on total number of cases passed (discussed in Section 4.4.1) and coverage of the design space. The coverage issue has already been discussed in the description of the methods. This can be verified by comparing plots of each input variable by each input variable (known as a scatter-plot matrix) and see if any of the design space is empty.

Figures 26 and 27 show the scatter-plots for the two methods compared here. The variables $u1$ through $u5$ are the control variables for the trajectory. A case is run at each of the dots shown in the plot. For every point on the plot, there may exist points behind it as well that represent points with the same values for the variables plotted on the graph but different values for the variables not in that particular graph. The fractional factorial design shown in Figure 26 covers the design space in a very structured way, while the Latin hypercube in Figure 27 seems random. At first glance it looks like the Latin hypercube actually covers much more of the design space, but it is important to remember these plots are 2-D representations of a multi-dimensional design space.

A natural question is whether all these points are required to cover the design space. It would be desirable to obtain the same information about the design space with fewer points. The fractional factorial is limited in terms of the flexibility it has with regard to the number of points in the design. Equation 13 defines what values

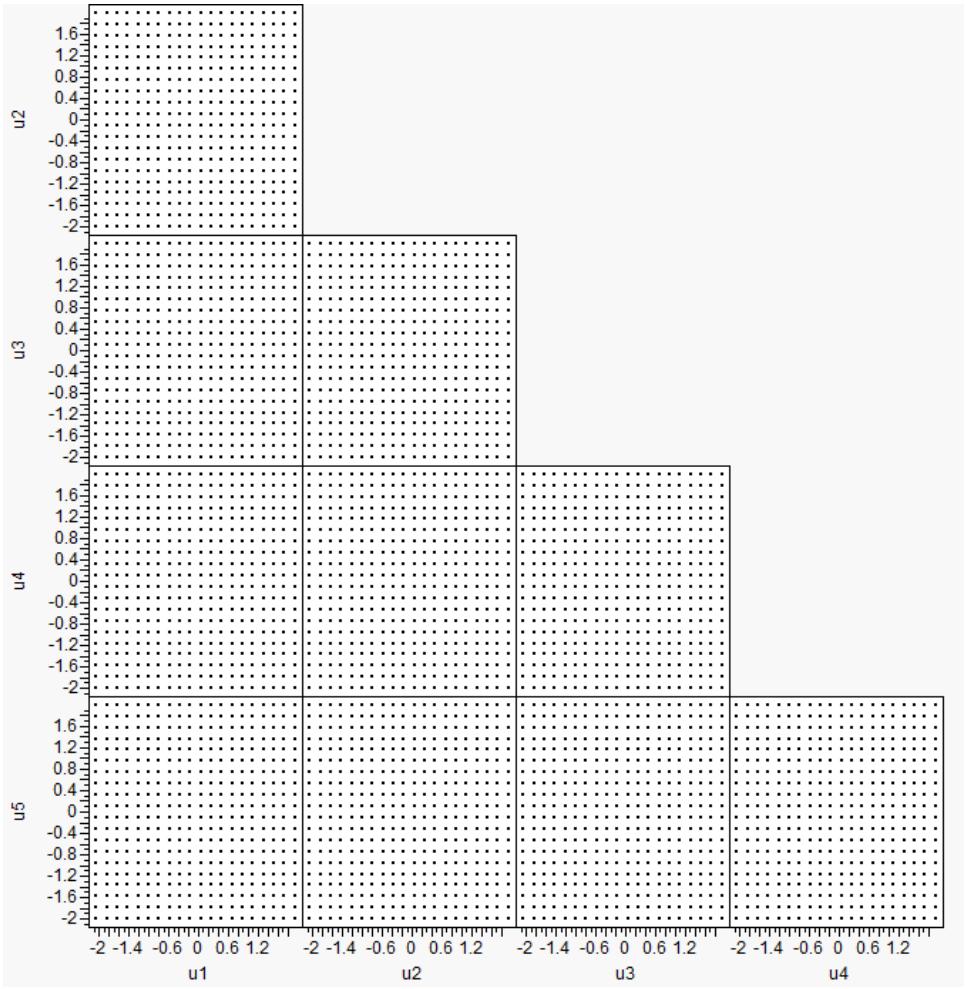


Figure 26: Scatter-plot for full factorial run

are available. Table 9 shows the number of experiments required for different number of levels.

Each of the full factorial designs in Table 9 was run for the example problem. Because the full factorial design was set up to cover the same range each time, the corners of the design space will be covered for each of the designs. However, the interior of the design space will be more and more sparse as the number of levels decreases. For example, Figure 28 shows the cases for a full factorial design with 5 levels compared to 20 levels used in Figure 26. The 5 level design is much more sparse.

The goal of the initial global search is to get information about what areas of the

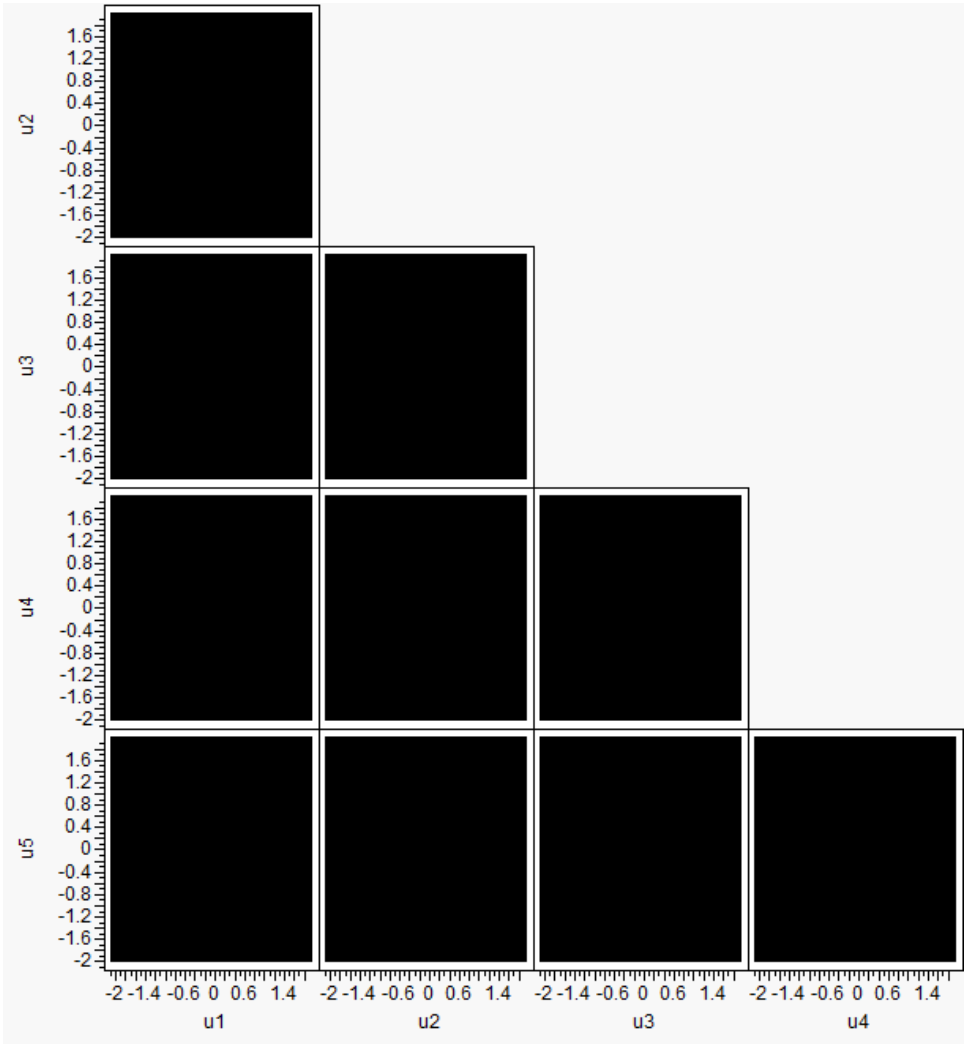


Figure 27: Scatter-plot for LHC run

design space are feasible. Recall, the design space is composed of the initial guesses on the control variables. There will be areas of the design space that do not yield cases that pass. Recall as well that cases that pass are cases that reach the trajectory termination criteria. With this in mind, the two experimental designs are compared.

4.4.3.2 Initialization Options Comparison

The goal of this section is to discuss how large the initial global search should be. Obviously, this will depend on the number of variables and the ranges of the variables. This will also depend on the type of trajectory that is being modeled. Determining

Table 9: Number of experiments for FF designs with 5 variables for different levels

# of Levels	# of Experiments
2	32
4	1024
5	3125
6	7776
7	16807
8	32768
9	59049
10	100000
11	161051
12	248832
15	759375
20	3200000

a rule set for any number of design variables or any trajectory is outside the scope of this problem. However, a general approach to determining whether more cases are needed or not can be developed.

The first step is consider where in the design space are the cases that passed. A scatter-plot of the feasible cases (Figure 29) clearly shows that not the entire designs space considered is feasible. For example, the control variable $u1$ only had values between -0.3 and 0.3 for any of the cases that passed.

Including values of $u1$ outside of the range from -0.3 to 0.3 in future cases is not advisable, as those cases will fail. Now it should be noted that because the design space is continuous and a finite amount of cases were run, it cannot be guaranteed that all cases not included in the range being discussed will fail. However, for the purposes of this study it will be assumed that the 6.4 *million* cases from the LHC and FF global initialization runs are a sufficient picture of the design space. Table 10 gives the design variable ranges for the feasible design space based on the LHC and FF global initialization runs.

An obvious way of cutting down on the number of failed cases is simply to decrease the range on the design variables. Figure 30 shows the scatter-plot for variables $u1$

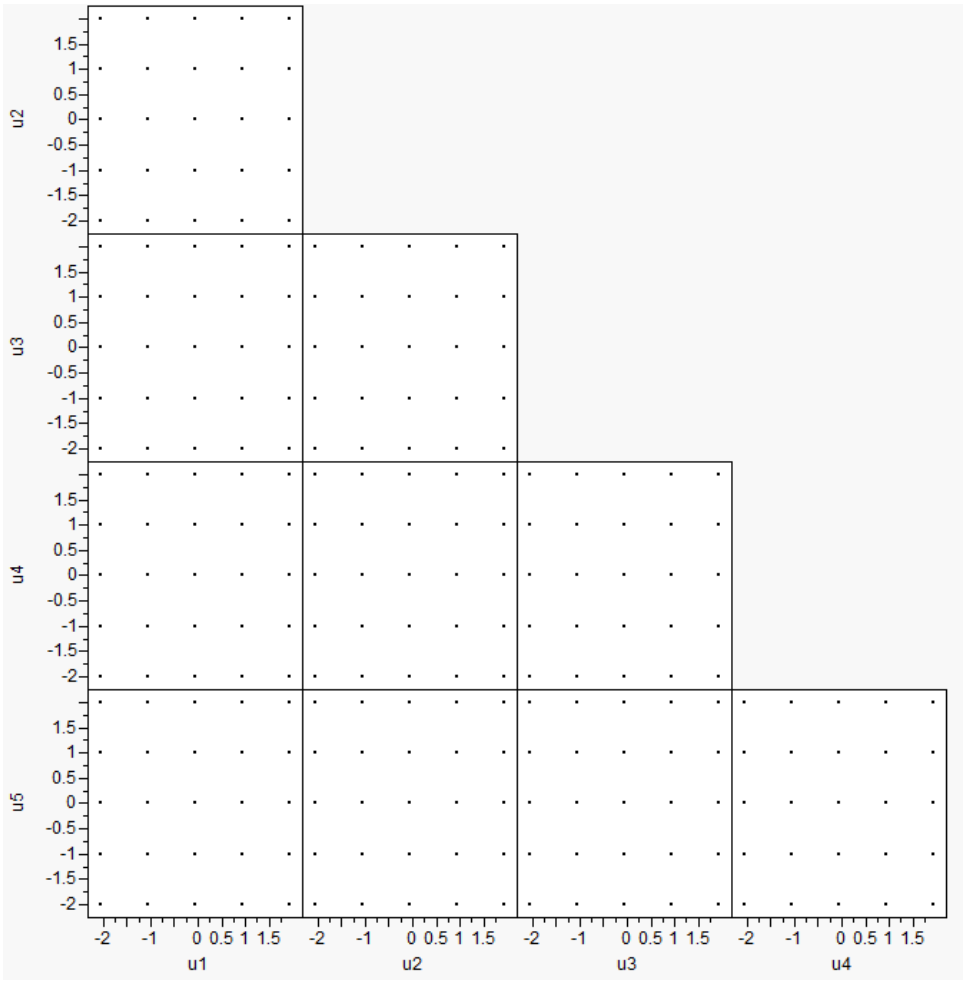


Figure 28: Scatter-plot for 5 level FF design

and $u4$ with red lines representing a new range for $u1$ to reduce the design space. This can be done for all the design variables, and the scatter-plot for all the variables can be seen in Figure 31. Clearly, there is much less infeasible space included in the ranges.

The question becomes how can the ranges be reduced without running 6.4 *million* cases. Ideally, a small global study would be performed, the design space reduced, and then a larger study would be performed to explore the feasible space in detail. It is impossible to determine *a priori* how many cases will be needed to ensure the whole feasible space is captured. However, if the feasible region stops expanding, even as more cases are run, it can be assumed that the ranges for the input variables have

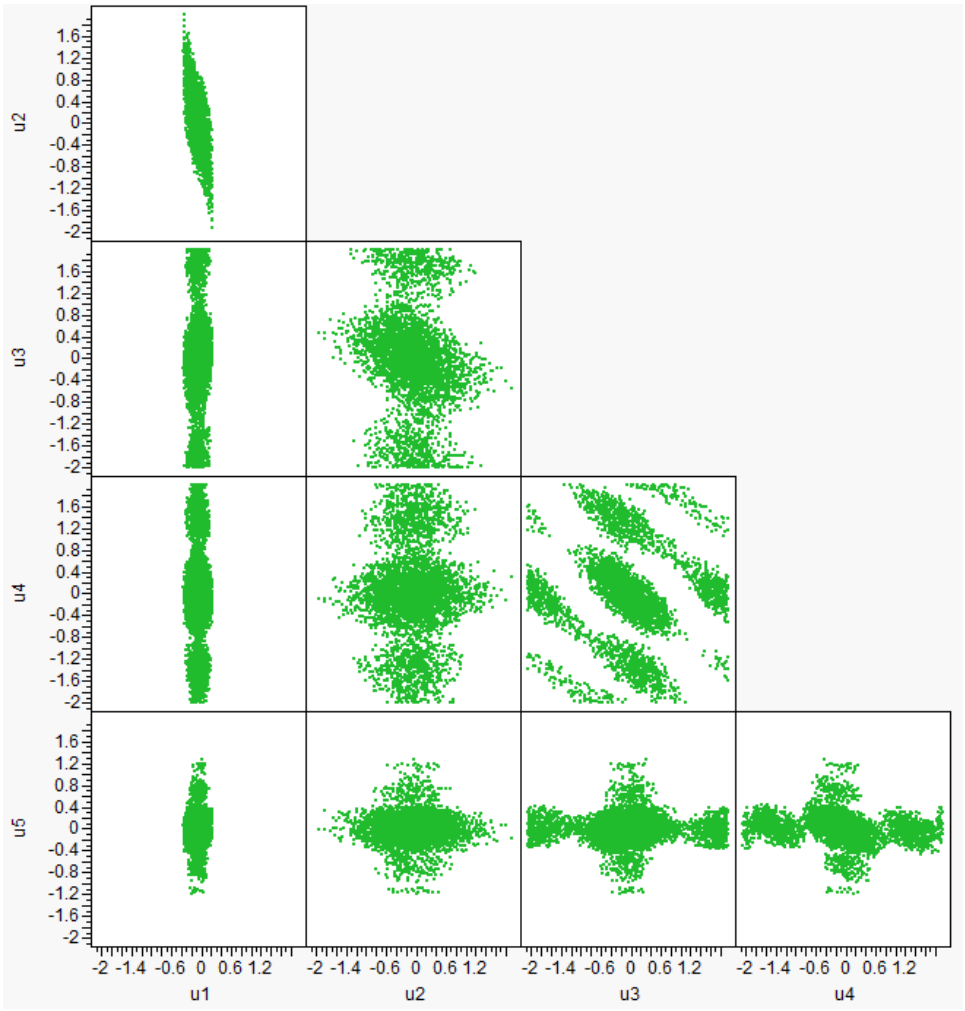


Figure 29: Scatter-plot for all cases that passed from LHC and FF designs

been determined.

To this end several FF and LHC designs were executed to compare how well defined the ranges were vs. the number of cases run. The LHC designs have the same number of cases as the FF designs to allow for comparison between the two types of designs. Because FF designs are have predefined number of cases, only those number of cases were initially considered. Table 11 shows the number of cases run, number of cases passed, and feasible design space range for each of the design variables for the FF designs. The ranges should converge as the number of cases goes up.

It is interesting to note that the FF design at 20 levels actually has a smaller range

Table 10: Feasible design variable range based on LHC and FF global initialization designs

Design Variable	u1	u2	u3	u4	u5
max	0.2963	2	2	2	1.2575
min	-0.2989	-2	-2	-2	-1.2417

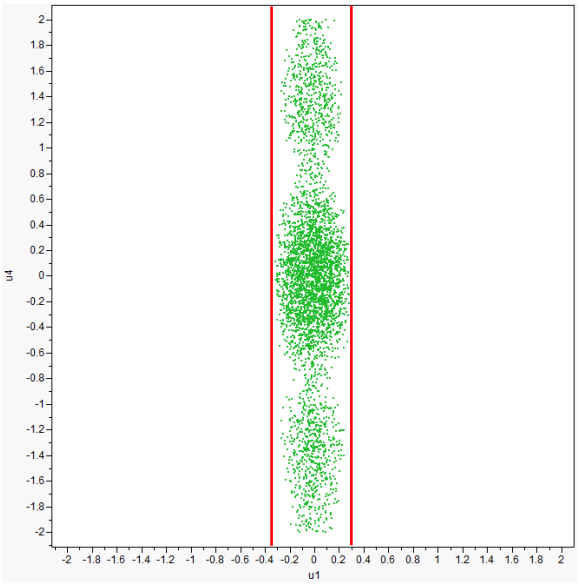


Figure 30: Scatter-plot example with reduced ranges to exclude infeasible design space

for some of the variables than the FF design at 15 levels. This can be for two reasons. The first is simply that the discrete values for the variables are preset. For example, for the overall range considered, -2 to 2, the value closest to 0.3 for the 20 level FF design is 0.315 and for the 15 level FF design is 0.286. If the feasible design space limit for the design variable $u1$ is around 0.3, as can be seen from Table 10, then a value of 0.315 will fail, while a value of 0.286 will not. The next value below 0.3 for the 20 level FF design is 0.105, which is the limit for $u1$ in the 20 level design. Because of the step values for the variables, it is difficult to precisely define the boundary between feasible and infeasible space. This is an important observation. A FF design alone is not enough to capture the boundaries of the feasible design space.

The second reason for the ranges becoming larger is related to the first. Whether the case fails or passes is not determined by the value of a single variable, but by all

Table 11: Passed cases and variable ranges for different FF designs

Levels		2	4	5	6	7	8	9	10	11	12	15	20
Cases		32	1024	3125	7776	16807	32768	59049	100000	161051	248832	759375	320000
Cases Passed		0	0	9	0	23	4	69	4	181	206	705	3213
Pass %		0	0	0.29	0	0.14	0.01	0.12	0.04	0.11	0.08	0.09	0.10
u1	max	n/a	n/a	0	n/a	0	0.2857	0	0.2222	0	0.1818	0.2857	0.1053
	min	n/a	n/a	0	n/a	0	-0.2857	0	-0.222	0	-0.1818	-0.2857	-0.1053
u2	max	n/a	n/a	0	n/a	0.6667	1.4286	1	1.556	0.8	1.2727	2	1.158
	min	n/a	n/a	0	n/a	-0.6667	-1.4286	-1	-1.556	-0.8	-1.2727	-2	-1.158
u3	max	n/a	n/a	2	n/a	2	0.2857	2	2	2	2	2	2
	min	n/a	n/a	-2	n/a	-2	-0.2857	-2	-2	-2	-2	-2	-2
u4	max	n/a	n/a	2	n/a	2	0.2857	2	1.556	2	2	2	2
	min	n/a	n/a	-2	n/a	-2	-0.2857	-2	-1.556	-2	-2	-2	-2
u5	max	n/a	n/a	0	n/a	0.6667	0.2857	0.5	0.6667	1.2	0.5454	1.1429	1.158
	min	n/a	n/a	0	n/a	-0.6667	-0.2857	-0.5	-0.6667	-1.2	-0.5454	-1.1429	-1.158

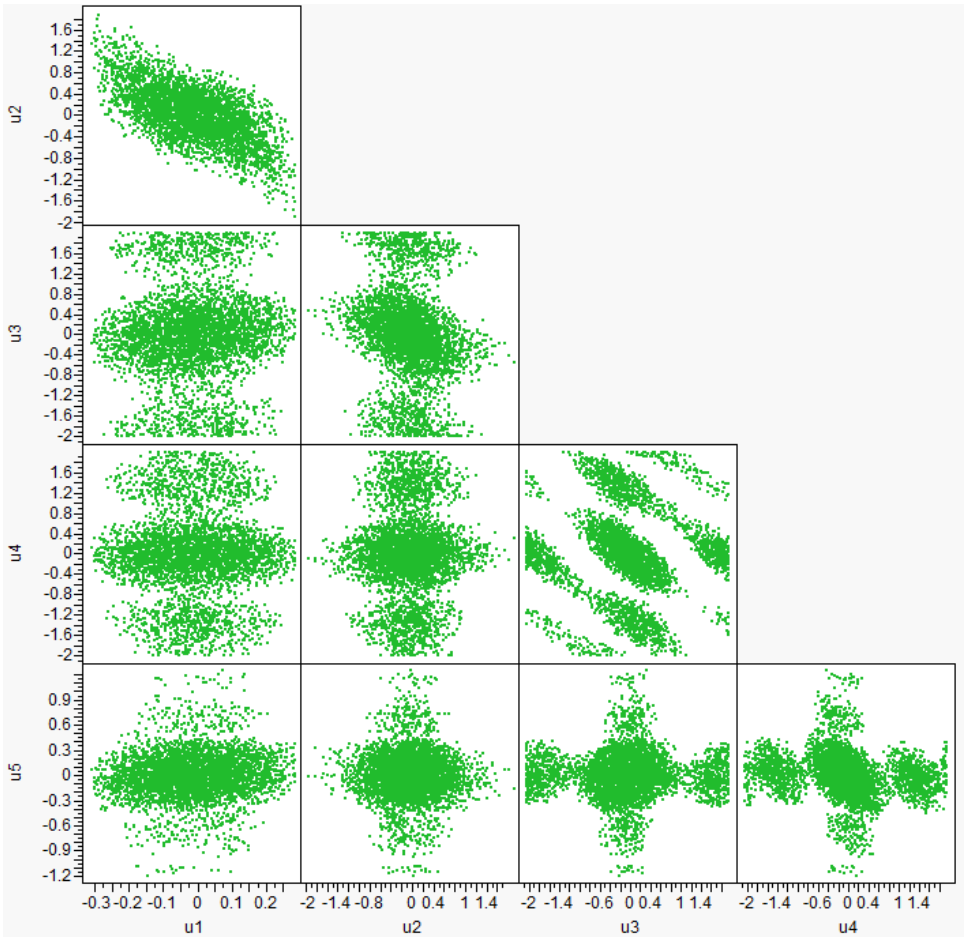


Figure 31: Scatter-plot with reduced ranges to exclude infeasible design space

the design variables. Therefore, because each FF design has different levels to which the variables are set, it is not impossible that the combination of variable settings lead to pockets of the design space that are not feasible. This effect may not be as noticeable with the higher level designs, but can be seen in FF designs with fewer levels. For example, a FF design with 4 levels has no passed cases. If 5 levels are used, 9 cases pass. If 6 levels are used, which would imply a better picture of the design space because higher resolution is used, no cases passed. This confirms the earlier conclusion that a FF design alone is not enough to capture the design space. The designer should not rely on an ability to guess the right number of levels to use to ensure the right variable values are sampled.

A similar analysis is done for the LHC designs. Table 12 shows the number of

feasible cases as well as the feasible design variable ranges for different LHC designs.

Unlike the FF designs, as the number of runs increases for the LHC designs, so does the number of cases passed. This is expected. Even though the LHC designs are random and more run cases will not always mean more passed cases, it is intuitive that the number of passed cases will be proportional to the total number of cases. This will be more and more evident as the number of cases run increases. The feasible design space is some fraction of the total design space, and if cases are randomly seeded, the expected result is that the percentage of cases passed would approximate the percentage of the design space that is feasible. Once the number of run cases gets to around $50,000$ the percentage of cases that will pass is consistently about 0.13%.

There are two important observations that should be noted in Table 12. The first is that few of the range limits for any of the variables are equal to the limit of the initial LHC design. This is due to how the design is set up. However, looking back to Table 11, after 16807 runs from a FF design, the ranges for $u3$ and $u4$ have reached their limits, and it can be determined that they do not need to be reduced. These limits are not determined until after over half a million cases are run using the LHC designs. It is clear then, that there is a benefit to running a FF design, because it does cover the edges of the design space. However, for determining boundaries that are not at the limit of the design space, LHC designs are more efficient.

The second observation is that the FF designs do not consistently capture the variable limits when those limits are inside the ranges sampled. Consider the control variable $u1$. The maximum value seen by a FF design for $u1$ is 0.2857. The maximum value seen by a LHC design for $u1$ is 0.2963. The FF factorial comes reasonably close. However, a 20 level FF design sees 0.1053 as the maximum value for $u1$ and a 12 level FF design sees 0.1818. Because of discrete nature of FF designs, it is difficult to capture variable limits when these limits do not correspond to one of the levels sampled.

Table 12: Passed cases and variable ranges for different LHC designs

Cases		32	1024	3125	7776	16807	32768	59049	100000	161051	248832	759375	320000
Cases Passed		0	0	3	7	20	37	77	126	201	325	985	4040
Pass %		0	0	0.10	0.09	0.12	0.11	0.13	0.13	0.12	0.13	0.13	0.13
u1	max	n/a	n/a	0.172	0.14162	0.26377	0.2678	0.27031	0.2631	0.2642	0.2619	0.29633	0.29362
	min	n/a	n/a	0.0312	-0.1847	-0.2227	-0.2509	-0.2607	-0.2778	-0.2932	-0.2888	-0.2914	-0.2989
u2	max	n/a	n/a	-0.4174	1.018	0.77869	1.1206	1.2396	1.7592	1.3734	1.8349	1.7462	1.9798
	min	n/a	n/a	-0.6672	-0.5037	-0.588	-1.2172	-1.4696	-1.6773	-1.6958	-1.4481	-1.9086	-1.9703
u3	max	n/a	n/a	0.03426	0.68322	1.9496	1.9107	1.9937	1.9751	1.9917	1.9933	1.9988	1.9988
	min	n/a	n/a	-1.799	-1.2667	-1.7626	-1.9872	-1.9648	-1.9901	-2	-1.9624	-1.9898	-1.9993
u4	max	n/a	n/a	0.5611	0.44587	1.2107	1.28	1.985	1.9969	1.9948	1.9742	1.9986	1.9988
	min	n/a	n/a	-0.1579	-1.4405	-1.8268	-1.7309	-1.6474	-1.7468	-1.9836	-1.9597	-1.997	-1.9926
u5	max	n/a	n/a	0.1404	0.0967	0.38899	0.4481	1.2293	1.0523	1.2373	0.9842	1.1296	1.2575
	min	n/a	n/a	-0.7168	-0.7733	-0.2507	-0.8016	-1.2417	-1.1616	-1.1912	-1.1421	-1.2329	-1.199

4.4.3.3 Initialization Conclusions

Based on the results from this test, it is concluded that the global search should be initialized using LHC designs. The FF design does allow for greater coverage at the edges of the design space. However, LHC designs will explore the interior of the design space. Based on the results showing in Tables 11 and 12, there are two possibilities for the design variables. The first is that the feasible region defined by the input variables reaches the limits imposed by the initial design. This is the case for variables $u3$ and $u4$. The second option is that there are limits in the design variables within the original limits defined by the experimental set up.

The results in Table 11 show that FF designs can find the variables that reach the limits imposed by the initial design more quickly than corresponding LHC designs. However, there is no way of knowing before hand which level to use for the FF design. For example, running an 8 or 10 level FF design would not give the correct limits for variable $u4$. Therefore LHC designs are used. The results in Table 12 show that around 250000 cases are required for the LHC designs to find the limits on the design variables. This is not an exact figure, and it will obviously change with different problems and different numbers of variables, but a strategy can be developed based on the results obtained. This is the number of cases where the percentage of passed cases becomes clearly defined and consistent. Once the percentage of passed cases is consistent, it is reasonable to conclude that the design space has been adequately sampled for the purposes here.

The conclusion for the initialization of the global search is to use a LHC design of 250000 cases. In addition, to confirm the limits have been found, several repetitions of a 50000 LHC design should be run. This is useful for several reasons. The first is that a 50000 case design is enough to find the limits, as can be seen in Table 12. The second is that LHC designs are random, and there is a chance that the limits will not be found by any of the designs. The more designs, however, the greater the chance

of finding the limits. The third reason is that by running separate designs, instead of a large single run, the limits can be compared. If there is no consistency at all in the limits, it may be useful to consider more runs. The last reason has to do with how the LHC designs are set up. More cases in a LHC leads to the LHC having a higher resolution, to use the term loosely. By having multiple different designs, the edges of the design space are sampled separately by the different designs.

This method for the global initialization is carried out and discussed subsequently in section 4.4.4.1.

This resulting method requires 500000 cases if 5 repetitions are used for the repeated LHC designs. This is more than an order of magnitude fewer cases than were originally run for the large LHC and FF designs. On a Dell Optiplex 790 machine with Intel i7 processor this run would take about 16 hrs. If this test is set up before leaving work on a typical work day, the cases will be completed when the work day begins the next day.

It should be noted that other designs may work better for this initialization set of cases. However, considering the scope of this document, these other designs have not been explored.

One additional note before this section on the global search initialization is completed is that of margins. It would not be completely amiss to add a margin on the limits found to help ensure the feasible region of the design space is captured. Adding a 5% or 10% margin on the design variable limits may be a good option, especially if few cases were run to find the limits. Testing what margin level is used is outside the scope of this problem, so for the purposes of this discussion a 10% margin is used.

4.4.4 Reducing the Design Space

The purpose of the global search initialization is to find the bounds and define the feasible region of the design space. As a reminder, the feasible region of the design space,

Table 13: Design variable ranges from proposed initial global run of 250000 cases

Design Variable	u1	u2	u3	u4	u5
max	0.2619	1.8349	1.9933	1.9742	0.9842
min	-0.2888	-1.4481	-1.9624	-1.9597	-1.1421

defined by the input variables, is the region where the trajectories meet the ending criteria. This translates directly into POST being able perform an optimization process on the trajectory. POST does not have a method of dealing with trajectories that do not meet the ending criteria.

Two methods (and several variants) to reduce the design space will be developed and tested. The first has already been discussed in the Section 4.4.3.2. The second method was introduced in Section 3.2.2.2. This method involves using Principal Component Analysis (PCA) to "align" the design space in such a way as to fit the feasible space as closely as possible, as depicted in Figure 15.

4.4.4.1 Reducing the Design Space by Reducing the Design Variable Ranges

The strategy presented in Section 4.4.3.3 was implemented assuming nothing was known about the design space. The first suggested run of about 250000 was be substituted with the 248832 case run since it was previously completed. The ranges based on this run are given in Table 13. The results from the 5 repetitions are given in Table 14. The results from all these experimental designs are combined, and a 10% margin is added. The limits are constrained to the original range of the experiments. The resulting ranges are given in Table 15.

The percent differences shown in Table 15 are positive if the 10% margin range encompasses the "truth" range, based on the 6.4 *million* cases from the initial LHC and FF designs. The only range that is not encompassed is the maximum value for $u5$. However, this is only off by 1.1%. Considering the fact that this is a stochastic problem dealing with a global search, this result is satisfactory.

Table 14: Design variable ranges from repeated LHC runs of 50000 cases

Repetition	Design Variable	u1	u2	u3	u4	u5
1	max	0.2561	1.4212	1.7267	1.9924	0.7077
	min	-0.2506	-1.3397	-1.8621	-1.9801	-1.1048
2	max	0.2689	1.1637	1.9464	1.9131	1.1305
	min	-0.2444	-1.2128	-1.8861	-1.9281	-0.8435
3	max	0.29576	1.0135	1.9049	1.7499	0.7146
	min	-0.2516	-1.9432	-1.9681	-1.8641	-0.7879
4	max	0.2546	1.291	1.9738	1.9396	0.6005
	min	-0.2845	-1.5674	-1.9914	-1.7282	-0.6437
5	max	0.2833	0.8905	1.9111	1.5343	1.0672
	min	-0.2801	-1.1829	-1.9869	-1.9323	-0.7316

Table 15: Design variable ranges from combined initialization results

	Design Variable	u1	u2	u3	u4	u5
Without Margin	max	0.2958	1.8349	1.9933	1.9924	1.1305
	min	-0.2888	-1.9432	-1.9914	-1.9801	-1.1421
10% Margin	max	0.3254	2	2	2	1.2436
	min	-0.3177	-2	-2	-2	-1.2563
"Truth" Limits	max	0.2963	2	2	2	1.2575
	min	-0.2989	-2	-2	-2	-1.2417
% Difference	max	9.8	0	0	0	-1.1
	min	6.3	0	0	0	1.2

A sanity check is performed as a way of confirming the proposed global initialization strategy. This check consists of comparing the scatter-plot shown in Figure 31 to the scatter-plot generated from the proposed initialization runs shown in Figure 32. As can be seen, the patterns are all clearly visible, even though the borders are somewhat less defined because fewer points were available. However, this does confirm that the design space is being captured with a lot fewer points. The question arises if even fewer points can be used to generate these patterns. This is a very interesting questions, but lies outside the scope of the problem. What is important is that the ranges found are satisfactory and a visual inspection of the feasible space serves as a confirmation that the methodology is adequate.

After the ranges were found a set of 100000 cases was created using a LHC design

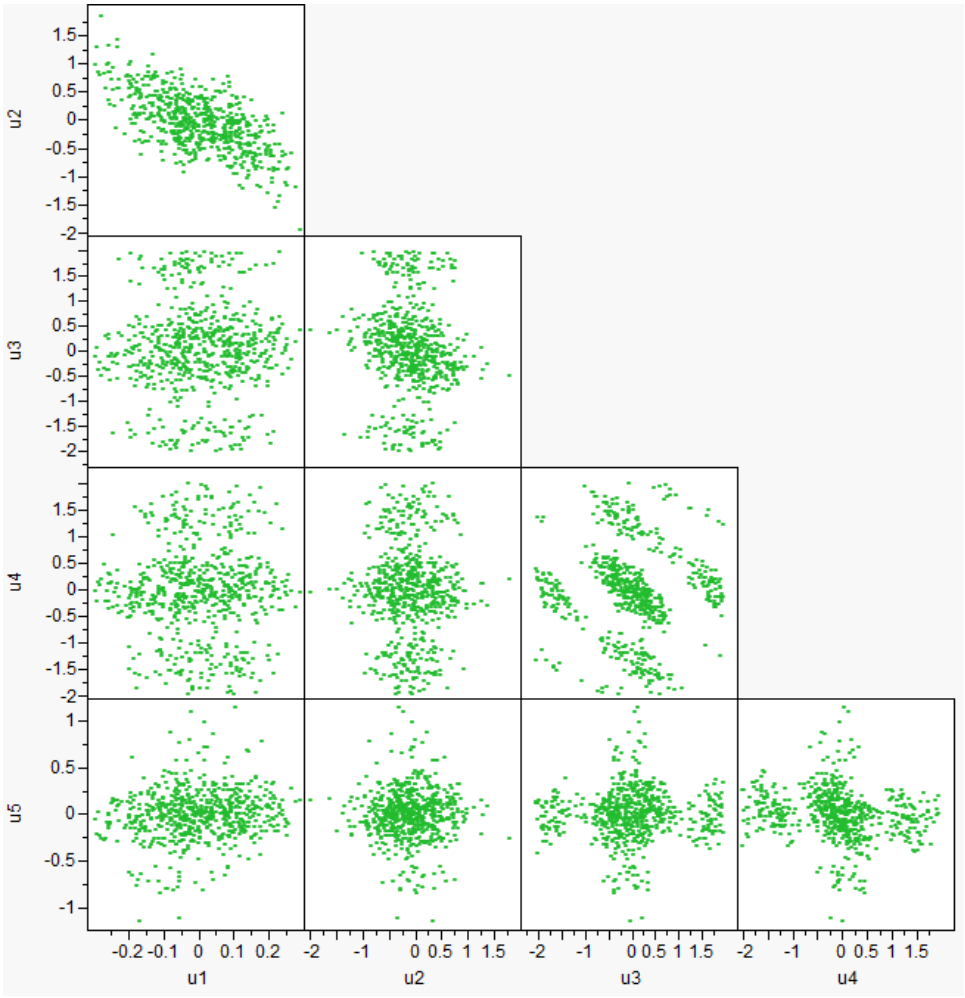


Figure 32: Scatter-plot of proposed global initialization results

with the new ranges. Running experiments with the new ranges should yield a much higher pass rate. It is important to note that the pass rate is not expected to be 100%. Referring back to Figure 31, it can be seen that even with the new ranges defined and the appropriate limits set, there are still areas with no feasible points. However these areas are much smaller than in Figure 30. The number of cases for this test was chosen based on the pass rates for the different experiments, seen in Table 12. In addition, the experiment was repeated to confirm the result.

The results indeed show that the pass rates increase significantly. Almost an order of magnitude improvement is seen. For the two designs, 1.20% and 1.18% of the cases passed. This may not seem like a good result, but recall that the best previous pass

rate was 0.13%.

Looking at Figure 31 it seems like a much higher number of cases should be feasible. After all more than 1.2% of the area in those plots is covered. The scatter-plots, however, can be misleading. While scatter-plots do give a good indication of where the feasible design is, it should be noted that there may exist multi-dimensional effects that are not captured. Within the cluster of points there may be regions of infeasible space that are masked by the points in front or behind it. Additionally, just because a point lies within the cluster of feasible points does not mean it will be feasible. The trajectory design space is very non-linear. The method here is based on never sampling an area where no points are feasible, not attempting to sample an area where all points are feasible.

Running another global search with reduced design variable ranges after the initial global search is not necessarily the best strategy for the overall methodology. Using a local search method at this point may be more effective. Before moving onto the integration of the local and global search methods, discussed in Section 4.5, other ways of reducing the design space are explored.

4.4.4.2 Reducing the Design Space using Principal Component Analysis

The previous section discussed the simplest way of reducing the design space with the data on hand. Another way is to employ Principal Component Analysis (PCA). PCA was discussed previously in Section 3.2.2.2.

A quick caveat is discussed before proceeding. The motivation of this section is not to find the best way to reduce the design space, or to compare all ways of reducing the design space. The motivation is to consider a way using PCA. This study will pave the path for future studies to develop and refine these and other methods related to trajectory design space exploration. The goal here is to show that this kind of trajectory search methodology is possible and beneficial and should be

explored further.

With that in mind, a discussion on the use of PCA to reduce the design space is included. The first step is to understand how PCA is useful. After performing PCA on the feasible points from the initial global search, the PCA inputs are uncorrelated. This means that picking a value for the first PCA variable gives you no information about the values of the other variables. This is not the case for the original design variables. Consider Figure 33. This is the same scatter-plot as Figure 31 except that the points with a value of variable $u1$ between about -0.2 and -0.3 are colored blue. This can be clearly seen the far left column of plots labeled $u1$. In column labeled $u2$, the points colored blue are on the right side of the plots. The divide between the blue and green points is not as clear as in the $u1$ column, but it is still there. This means that if a $u1$ value between -0.2 and -0.3 is selected, then the $u2$ value must fall somewhere on the right side of the $u2$ range for the point to be feasible. Knowing the value of $u1$ gives information about the value of $u2$.

This means there is correlation between variables $u1$ and $u2$; i.e. $u1$ and $u2$ are not independent [70]. The Pearson product-moment correlation is a measure of how much a variable depends on another [41]. There are other ways to measure correlation, but those methods are not discussed or used here. Figure 34 gives the correlation between the 5 input variables. If variable values are chosen at random between the limit ranges of the design variables, this will not take into account the correlation between the variables. This is essentially what was done on Section 4.4.4.1.

A better method would be to apply PCA and generate an uncorrelated space. Then a LHC or Monte Carlo can be generated using the uncorrelated variables, and these points can be translated back to the original design variables. This enables choosing design variable points at random without having to take into account the correlations. The correlations are not ignored, they just do not exist in the design space generated by applying PCA. Note that PCA only operates on linear correlations.

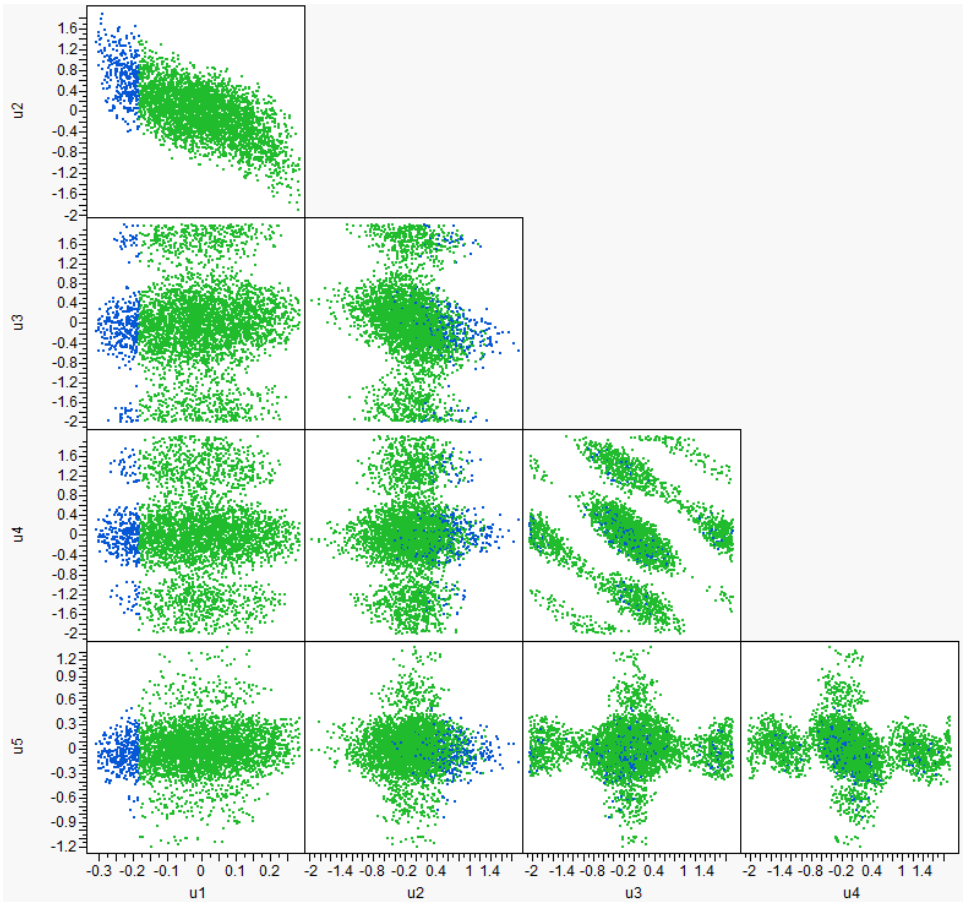


Figure 33: Scatter-plot with specified range of $u1$ colored blue

	u1	u2	u3	u4	u5
u1	1.0000	-0.6538	0.0415	-0.0114	0.0986
u2	-0.6538	1.0000	-0.1621	0.0286	-0.0075
u3	0.0415	-0.1621	1.0000	-0.0487	0.0154
u4	-0.0114	0.0286	-0.0487	1.0000	-0.1818
u5	0.0986	-0.0075	0.0154	-0.1818	1.0000

Figure 34: Correlation values for design variables

While non-linear patterns may exist in the feasible design space, they are not dealt with in this discussion.

Using the data from the method proposed in Section 4.4.3.3, this strategy employing PCA was applied. The first step was to perform PCA and this was done using MATLAB. MATLAB is a mathematical software package that includes a function (*princomp*) to perform PCA [5]. Without going into much detail, the principal components were found and a mapping function was created to transform the design variables into principal component variables and vice versa. For more information how exactly how this was done the reader is referred to Jolliffe [42]. A LHC design was then be created using the ranges from the principal components. After the LHC design was mapped back to the original design space, the result was an experimental design that took into account the correlations of the feasible region of the original design.

Two different LHC designs were created, each of 100000. Again, the number of cases was chosen based on the pass rates for the different experiments, seen in Table 12. When the principal component design space was created, there exist corners of the new space that do not have any feasible points in them. Figure 35 is the scatterplot of the feasible points in the principal component space from the proposed global initialization runs. While patterns can still be seen between some of the variables, Prin3 and Prin4 for example, most of the plots approximate points distributed in a circle. In addition, the correlation between all of these points is 0. However, returning to the original point of this paragraph, there are corners of the principal component design space that do not have any points in them. This means if an experiment design is set up within the ranges of the points given, there will exist points where no feasible points are. When this experiment design is mapped back to the original design space, some of the points will be outside the original ranges of the feasible points in the original design space.

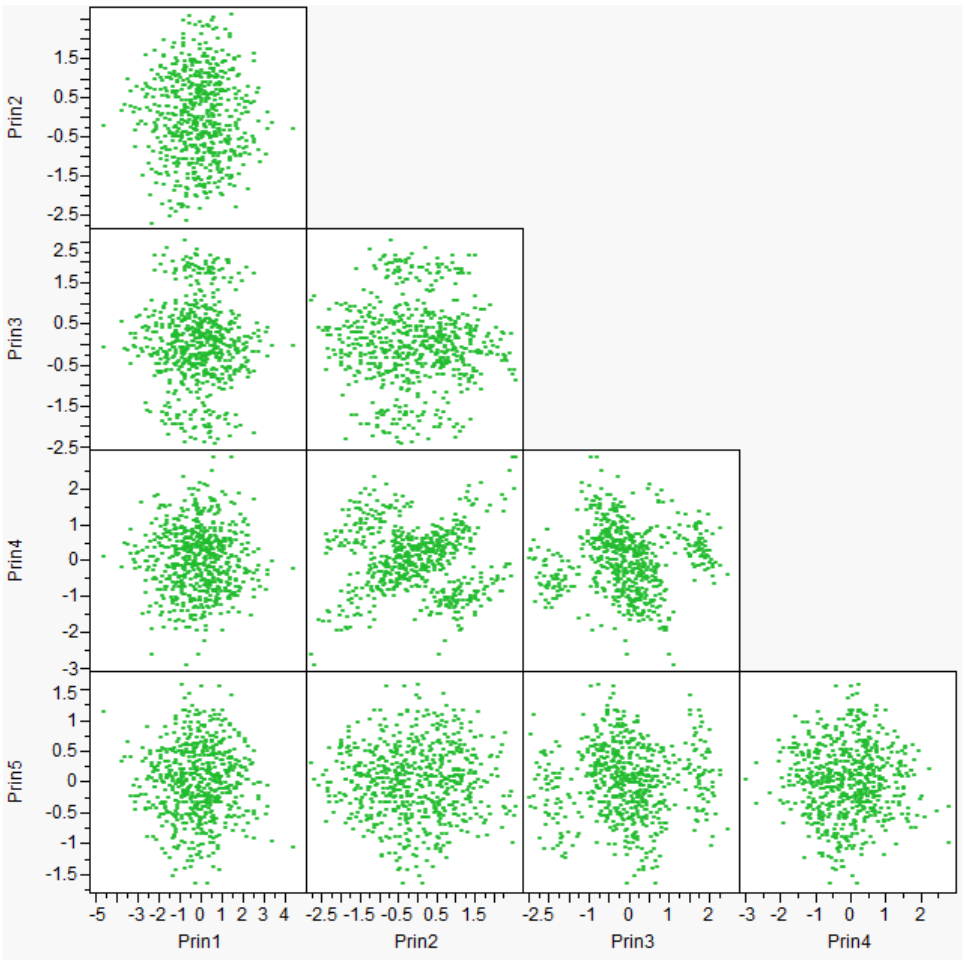


Figure 35: Scatter-plot of feasible points in principal component space

Figure 36 shows the LHC design in the original design space. The ranges on design variable $u1$, for example, are from about -0.55 to 0.5 instead of about -0.3 to 0.3. The ranges for the other variables increase as well.

Even with this increase in the design ranges, some of the corners in the original space that had no feasible points are excluded using this method. Referring back to Figure 31 the shapes of the patterns are matched more closely than a LHC design created in the original design space, an example of which can be seen in Figure 27.

The results of running the LHC designs designed using this method yield a pass rate of 3.25% and 3.33%. It is obvious that this method significantly increases the pass rate more than merely reducing the ranges in the original design space. However,

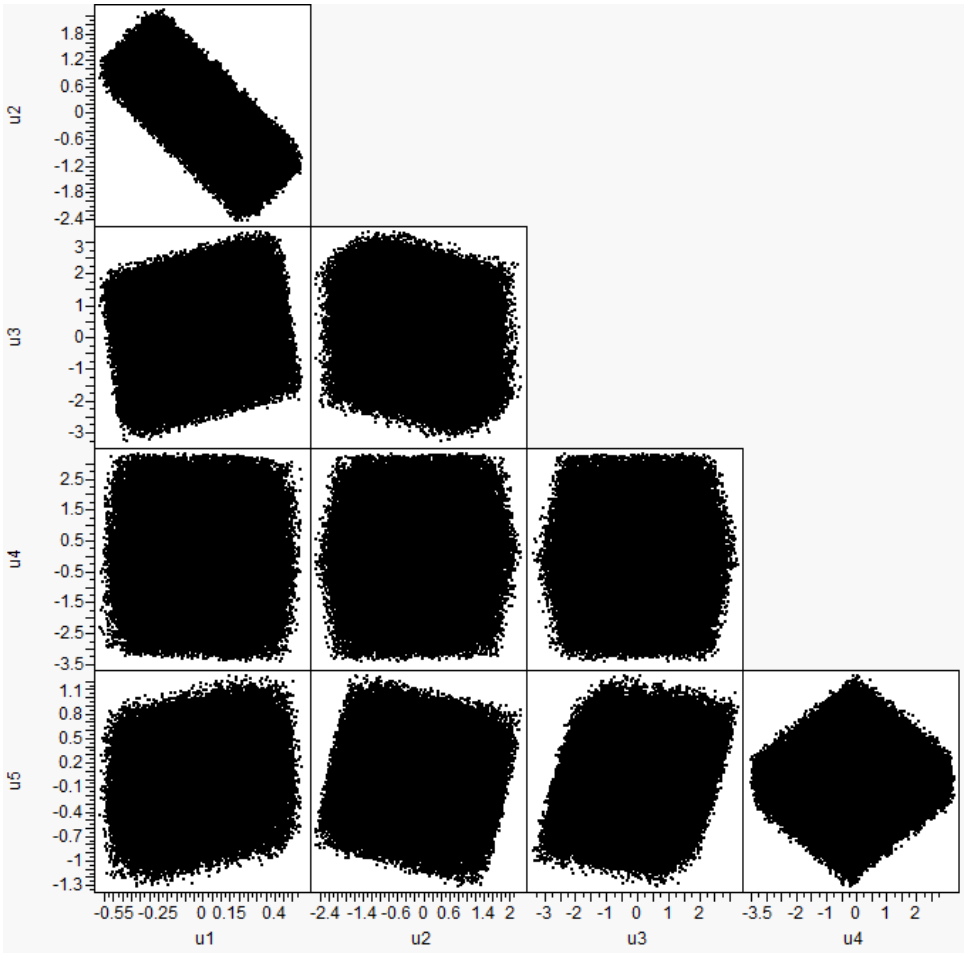


Figure 36: Scatter-plot of LHC design generated in principal component space and mapped to original space

a better method exists to reduce the design space.

As discussed earlier the LHC design created in the principal component space and mapped backed to the original leads to ranges of the original design variables outside what is known to be feasible. If the LHC design cases are scaled to fit within the known ranges for the feasible region in the original design space, a higher percentage of the cases will be feasible. The LHC design looks exactly like the scatter-plot shown in Figure 36 except that the ranges are equal to the 10% margin values shown in Table 15.

Using this strategy does have the disadvantage that some of the corners of the feasible space are not well covered. After running all these cases, the percentage

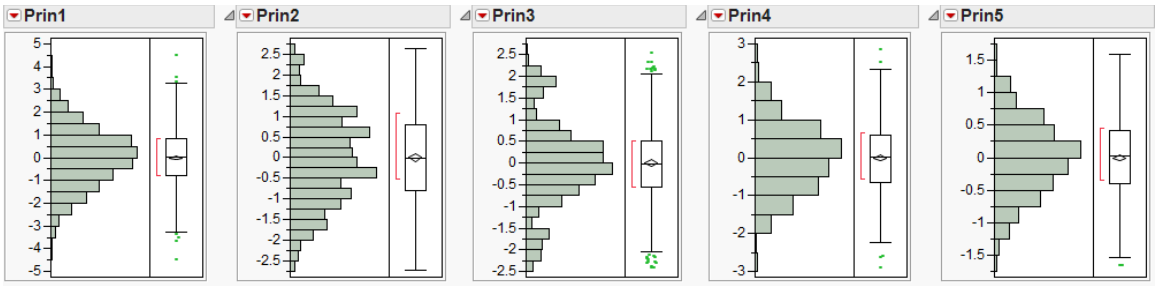


Figure 37: Distributions of feasible points in the PCA design space

of passed cases increases. Again, because these designs are stochastic, 2 different designs were created to ensure the result was not an outlier. The resulting pass rates are 9.62% and 9.26%. This is almost a 2 order of magnitude increase from the original pass rates, seen in Tables 11 and 12.

There is another strategy that can be implemented using the principal component space. Instead of using a LHC design in the principal component space, a Monte Carlo can be generated using the distributions of the principal components. Consider Figure 35. These points represent the feasible region of the design space, and are visualized in the principal component space. It is obvious that the distributions of these variables are not uniform. In fact, Figure 37 shows the distributions, and they are far from uniform.

Instead of applying a uniform distribution then, as a LHC design essentially does, a Monte Carlo can be generated using the distributions of the principal components. Without going into much detail, an open-source MATLAB script based on MATLAB's function to fit distributions (*fitdist* [5]) was used to find distributions that approximate the data representing the feasible points [62]. No discussion is included here as to what is the best way to match distributions or how is "best" measured when it comes to comparing distribution fits. That is not the goal of this study. The goal of this study is to show that this approach, i.e. a global search and design space reduction, is useful in trajectory optimization. This study will hopefully pave the way for future studies to optimize different ways to do this.

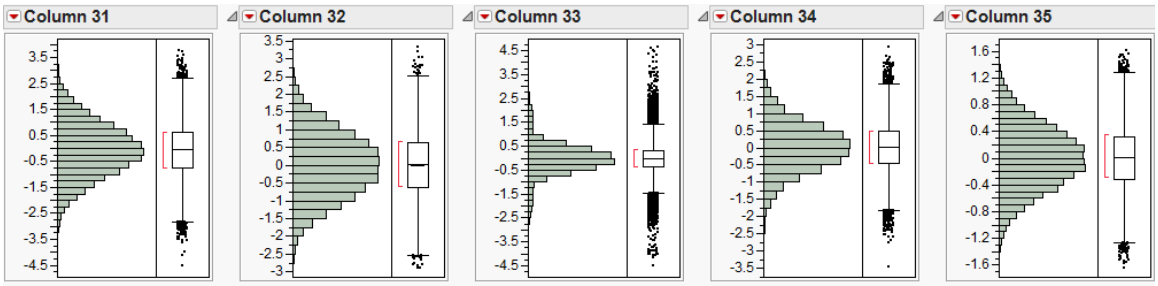


Figure 38: Approximated distributions of feasible points in the PCA design space

Once a distribution to match the data was found, points were randomly seeded using that distribution, and an experimental design was created. In this case the designs are not LHC designs, instead they are Monte Carlo designs using specific distributions based on the feasible points data. The distributions of this Monte Carlo design is shown in Figure 38. The distributions do not exactly match those shown in Figure 37, but they are closer than uniform distributions. Mapping these points back to the original design space shows that the patterns in the scatter-plot in Figure 39 reasonably match the patterns seen in the feasible design space. Obviously the matches are not perfect, but it is the best approximation seen thus far.

Two such Monte Carlo designs were created and run, and the results yielded a 23.30% pass rate both times. This is a dramatic improvement on the original 0.13% pass rates seen in the original runs. Almost 200 times more cases pass when this methodology is employed.

It is concluded then that this design space reduction strategy, employing PCA and matching distributions, should be used to generate random sets of experiments if feasible points are desired.

4.4.5 Global Search Summary

In summary, there were 2 main answers to this Research Question. The first is that the global search should be initialized using LHC designs; 1 large design combined with several smaller repeated designs to try to ensure that the design space is captured.

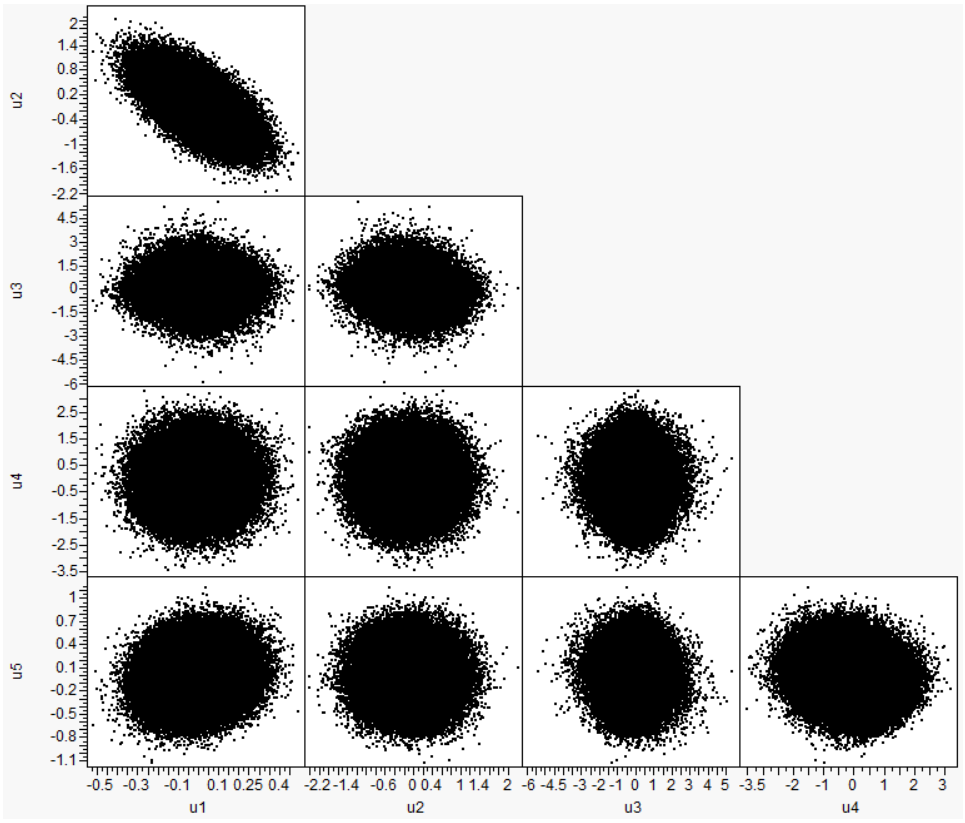


Figure 39: MC points generated using approximated distributions of feasible points in the PCA space

After this, PCA should be used on the feasible points to reduce the design space. At this point there exists a design space where a larger percentage of cases will be feasible, and the information can be passed onto the local optimization part of the methodology.

4.5 Experiment 4: Integrating Global Search and Local Optimization

The global search was focused on finding feasible regions of the design space. Global optimization methods exist to fine tune candidate solutions, but in general local optimizations methods are better suited to this task [16]. For this reason, a local optimizer is included in the methodology. When performing local optimization the goal is no longer to find feasible space, but to find the best performance. The feasible

space has been found, and now in the feasible space, the best solution is desired. The reason the feasible space had to be determined first was because performing local optimization on the entire design space is computationally expensive. In fact running the same set of cases with and without the optimization option in POST takes about 40% longer, even when the time it took to optimize the cases that were feasible was taken into account. It is expected that a trajectory propagation tool can be developed with the express purpose of determining feasibility, and the time difference would be even greater. However, this effort is outside the scope of this project.

Once the design space has been reduced, local optimization can be performed. This section addresses Research Question 4.

Research Question 4 - Will using local optimization on the results from the global search yield better solutions?

Research Question 4 will be answered with two experiments. The first is whether to create a new experiment design and run a set number of cases or simply use the passed cases from the global search. These two designs will be compared not on number of passed cases, but based on the performance metric, in this case weight.

The second will address the issue of iterations. After local optimization is run it may be worth while to reduce the design space again based on the cases that passed and were able to be optimized. The concept of a passed case will be revisited here to avoid confusion. A case that passed is a case that reached the ending criteria, therefore allowing it to be operated on by the local optimization methods available in POST. An optimized case is a case that, after being operated on by POST, achieves the desired orbit while minimizing or maximizing a specified parameter, in this case maximizing the weight it ends with. After all the cases have been optimized the

design space can be reduced using the optimized cases, instead of cases that merely pass. Running a design of experiments in this space may yield better solutions. For this problem about 53% of the feasible cases are optimized by POST, meaning they meet the constraints and have converged [56].

4.5.1 Experimental Setup for Local Optimization

A comparison is made here between running the cases that were feasible from the global initialization run and running a random set of cases set up using the PCA method, described in Section 4.4.4.2. Unlike the global search, the goal is performance, not a pass rate. Having said that, it is logical to assume that higher performance would be achieved with a higher number of optimized cases, so a high pass rate is still desirable. Using the PCA method, about 23% of the cases are feasible, and after running all the feasible cases, only 53% of the cases were optimized. Therefore, if a new design is set up using the PCA method for local optimization, it can be expected that about 12% of the cases will be optimized.

A histogram of the results from optimizing all of the feasible cases from the global initialization is shown in Figure 40. Of the 632 cases 340 were able to be optimized, hence the 53% optimization rate of feasible cases. As can be seen, all the cases show negative propellant remaining. The example problem was set up so that it would be difficult to find the best solution. In this case, the best solution was $-1888\,lbs$ of propellant remaining. This means that of the $44974\,lbs$ of propellant in the upper stage, the trajectory required $1888lbs$ more. This is about 4% more fuel than is quoted in the Delta IV User's Guide [4]. Even though the propellant required does not match perfectly, the method for local optimization can still be studied and developed.

A comparison is made between running the feasible cases and setting up a new set of experiments using the PCA method. There were 632 feasible cases after the global initialization. Ten new sets of experiments, each of 632 cases, were set up

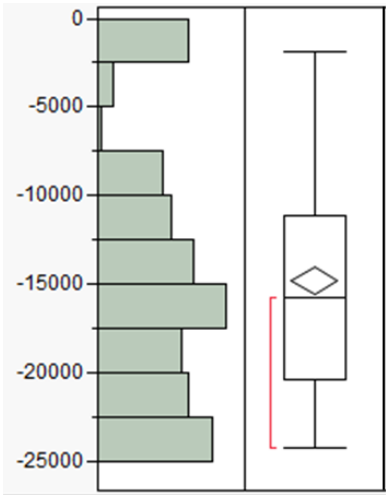


Figure 40: Histogram of propellant remaining after optimization of the feasible cases from the global initialization

using the PCA method. As the method is random in nature, ten repetitions were used to ensure consistency. The optimized rates are somewhat inconsistent, but that is expected given the low number of cases. The average optimized rate is 12.7%, which is as expected. The histograms of the ten repeated experiment sets are included in Appendix A and it can be seen that the histograms are all very similar.

It is interesting to note that the best solution did come from running the already feasible cases. However, it is important to recall that the set of feasible cases had 340 optimized cases to choose from, whereas the repeated sets had only about 80. There is no reason to believe that running the feasible cases vs. a new set would make any difference other than that as the number of optimized cases to choose from increases, better solutions will result.

To show this, two sets of 1000 cases were set up and optimized. The optimized rate for these were 10.6% and 12.2% and the best solutions were -1874 *lbs* and -1955.4 *lbs* respectively. These two results are different, both in optimized rate and the best solutions. In addition, the set with fewer optimized cases found the better results. It shows that this process is random in nature, and repetitions are required to increase the probability of finding the best result.

Table 16: Results from 10 repetitions of experimental setups for local optimization using the PCA method

Experiment	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	69	10.9	-1897.7
2	85	13.4	-1888.4
3	87	13.8	-1901.9
4	86	13.6	-1888.7
5	87	13.8	-1891.3
6	80	12.7	-1904.9
7	95	15.0	-1894.0
8	75	11.9	-1910.4
9	70	11.1	-1888.4
10	68	10.8	-1892.9

This same experiment was done again with 10000 cases. In this case the optimized rates were 12.2% for both and the best results were -1888.3 *lbs* and -1887.2 *lbs*. These results are definitely more consistent, but are not the best found. For all these experiment sets, the histograms are similar and are shown Appendix A.

From these results, the recommendation is to run as many cases as possible within the feasible space. While this method does not guarantee the best solution, it does give the best change. Because the experimental setup method is completely random (as opposed to a LHC setup), there is no difference between setting up 1 design of $10,000$ or 10 sets of 1000.

Another method for the local optimization is presented and tested in the next section to see if better solutions are found.

4.5.2 Reducing the Optimized Space using Principal Component Analysis

A data set of optimized trajectory control variables is available after running the feasible cases from the global initialization. The optimized control variables can be compared to the input control variables for all the optimized cases. This is shown in Figure 41, where the green points are optimized and the black points are from the experiment set up.

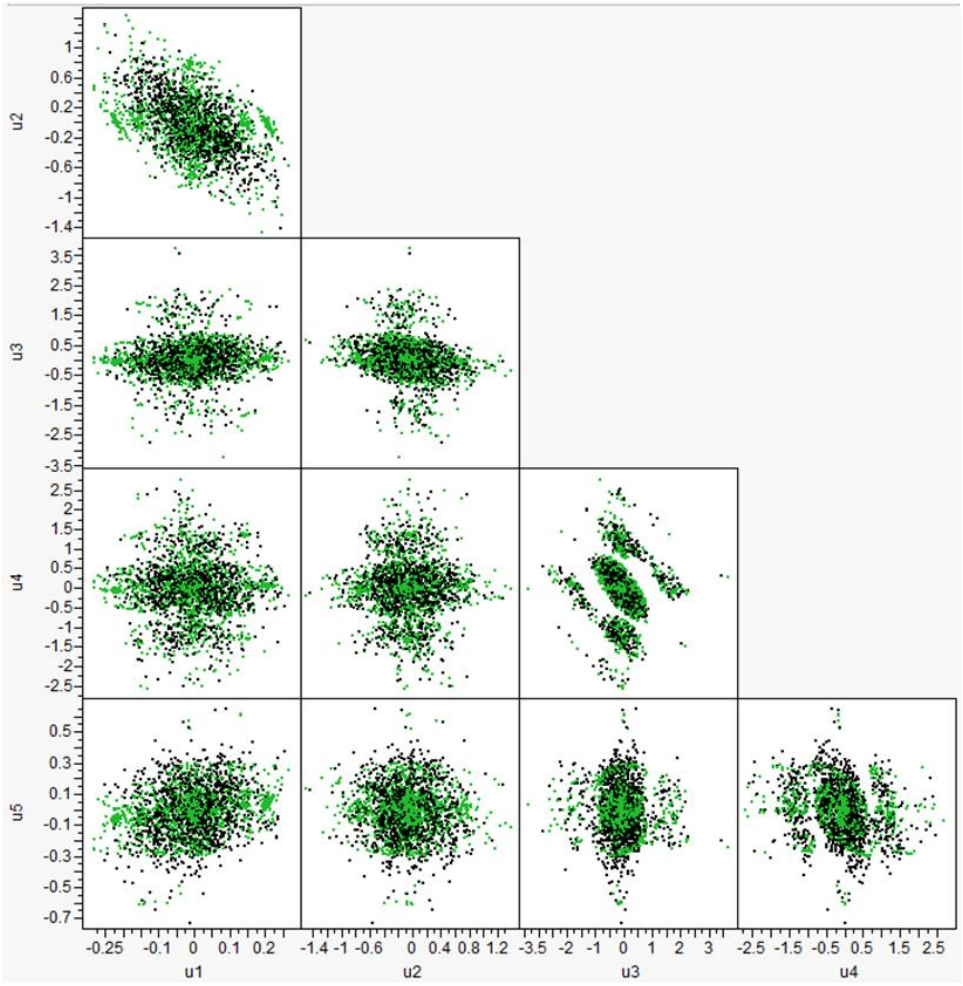


Figure 41: Scatter-plot of input control variables with overlaid optimized control variables

It can be seen that there is very little difference between input space and the output space. In other words, there would be little or no benefit in using information from the "optimized" space vs. the "guess" space to develop a new space to sample from to try to find better solutions. Recall that the "guess" space already exists and is used to generate the cases.

However, the goal here is performance. Not all the optimized solutions are equal. Some have better performance than others. Consider Figure 42. This plot shows all the optimized points for one of the 10000 case experimental setups from Section 4.5.1. A color scale is applied to the cases based on propellant remaining, where blue

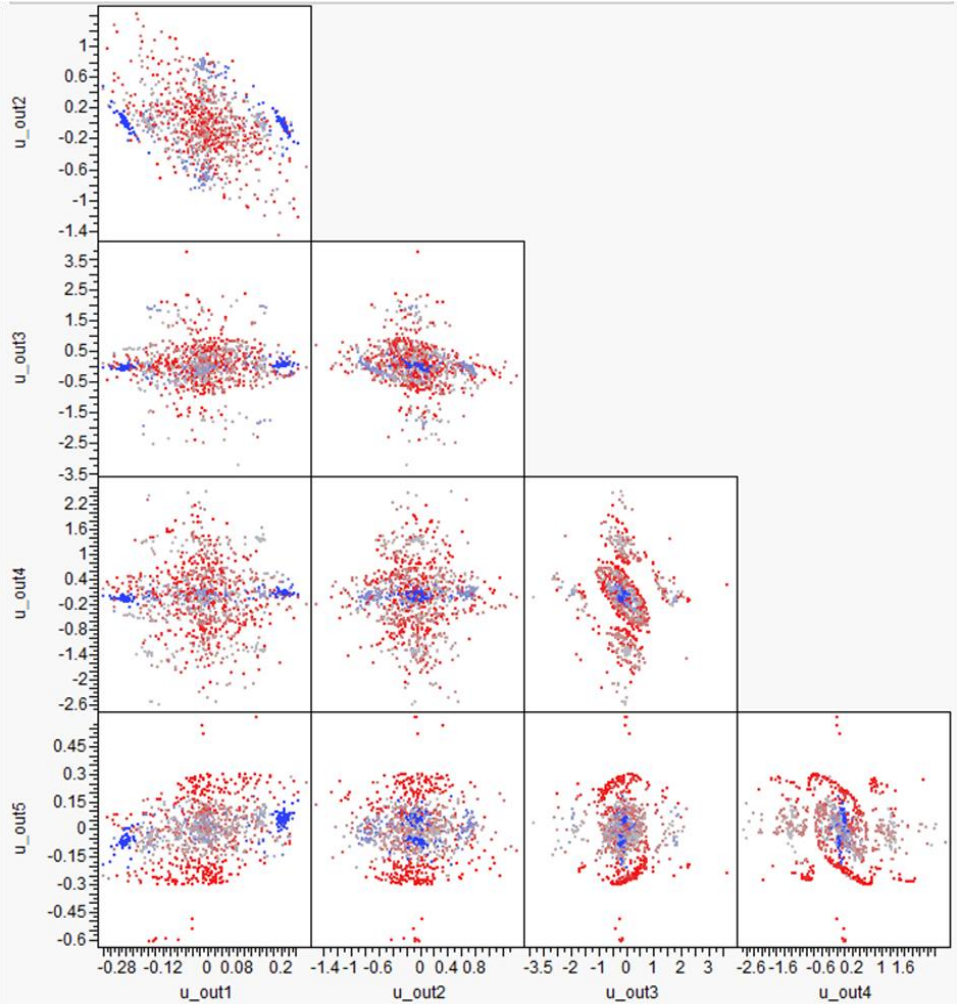


Figure 42: Scatter-plot of output control variables with color scale representing propellant remaining values

represents highest values and red represents lowest vales. The goal is to have the most propellant remaining possible.

There is obviously a pattern where the best performing cases are appearing. For the control variable u_out4, for example, there is a very thin band around about -0.2 where all the best performing cases lie. Earlier in this study, PCA was performed on feasible cases to create a design space that would increase the number of feasible cases in a design set. The same method can be applied to any set of cases. If this method is applied to the top 10% (for example) of the cases based on performance, the odds of finding better solutions may be increased. Here the criteria for selecting

the cases is performance rather than feasibility.

The actual percentage does not matter. The idea is to generate a design based on the best performing cases, whether this is the top 1% or the top 25% is irrelevant, as long as there are enough cases to perform the PCA. Determining the number of cases required is not discussed here. The reason, as will be seen in the following paragraphs, is that this method works. When using PCA to define the feasible space, it was important to completely capture the feasible space. If there were regions of the feasible space that were not captured, the best solution may be overlooked. In this case, however, the feasible region has already been sampled and those sample points have been optimized. The goal is to continue sampling to find the best possible solution. It is acceptable at this point to cut off regions of the feasible space. Using PCA in this case is merely trying to intelligently seed points where it is most likely to find the best solutions.

This was tested by taking the top 10%, again this value was chosen arbitrarily, and applying the method described in Section 4.4.4.2. As a review, PCA was performed on the cases of interest and a transformed design space was generated. The distributions of the PCA variables were approximated using the data. Then random points were selected using the approximate distributions to generate Monte Carlo sets that lie in the desired space. Recall the desired space is the region defined by the top 10% of cases based on propellant remaining after optimization. Because this process is inherently random, repetitions are used to ensure consistency. Four sets are run, two of 1000 cases and two of 10000 cases.

The results of the experimental sets are shown in Table 17. The first observation comes from examining a histogram of the results. Figure 43 shows a histogram for the first case set, which is representative of the other histograms as well (included in Appendix A). Obviously most of the cases are on the upper end of the histogram. In fact, for the first case set, over 90% of the case end their trajectories with -2200

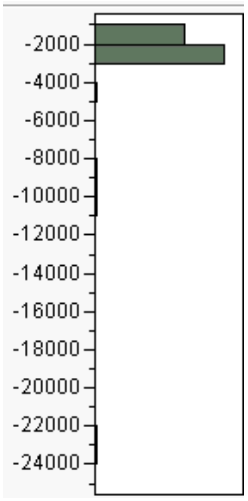


Figure 43: Histogram of propellant remaining after optimization of the "top-performing" cases after initial optimization run

lbs or greater. Compare this with the histogram shown in Figure 40, where only about 10% of the cases are -2200 *lbs* or greater. It is clear, then, that more cases are performing better using this method.

The second observation is that the results for the 1000 case sets are not consistent, either in optimized rate or in the best solution. The 10000 case sets, sets 3 and 4, are consistent in terms of optimized rate, but not in terms of the best result. Even after running 2 sets of 10000 cases in the most promising areas in terms of performance, different best solutions are found. However, they are better than anything found so far.

Table 17: Results of experimental sets created using the best performing cases after optimization

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	623	62.3	-1874.4
2	1000	429	42.9	-1867.1
3	10000	4052	40.5	-1825.3
4	10000	4002	40.0	-1768.8

To see if it is possible to reach consistent results, two more case sets were run, each of 100000 cases. Whether or not consistent results are achieved, it is clear that taking

into account the best performing cases to sample more cases in promising regions is of great benefit.

Running the two sets of 100000 yielded significantly better results. The optimized rates were very similar at 40.5% for both of them. The best solutions were $-1730.2 lbs$ and $-1718.5 lbs$. This is a significant improvement from the results in Table 17. This improvement comes at the cost of an order of magnitude higher number of cases. This is an expected result. The more cases are run, the more likely to find the actual global optimum. However, the return on investment decreases as the number of cases increases. Running 1000000 cases will likely give a better result, but maybe only a $20 lbs$ improvement instead of the $50 lbs$ improvement seen here.

4.5.3 Local Optimization Integration Summary

The results of the local optimization integration tests show that using the feasible cases from the global initialization is a good idea only because the cases have already been found to be feasible. More experimental sets can be run to get a better picture of the design space using the PCA methodology described in Section 4.4.4.2. After a set of optimized cases is found, the best performing of these can be selected. Performing PCA on these cases will isolate a region of high performance and increase the chances of finding the best solution. The number of cases run at this point is determined by how much performance is desired and how much computational expense is acceptable.

4.6 Applying the Methodology to Generic Optimization Problems

The methodology presented in Sections 4.4 and 4.5 can be tested using benchmark optimization problems to see if this method is applicable to generic optimization problems. Many such problems exist and a list of these problems has been compiled by Jamil et al [40]. The methodology will be tested using two of these problems. The first is known as the helical valley function.

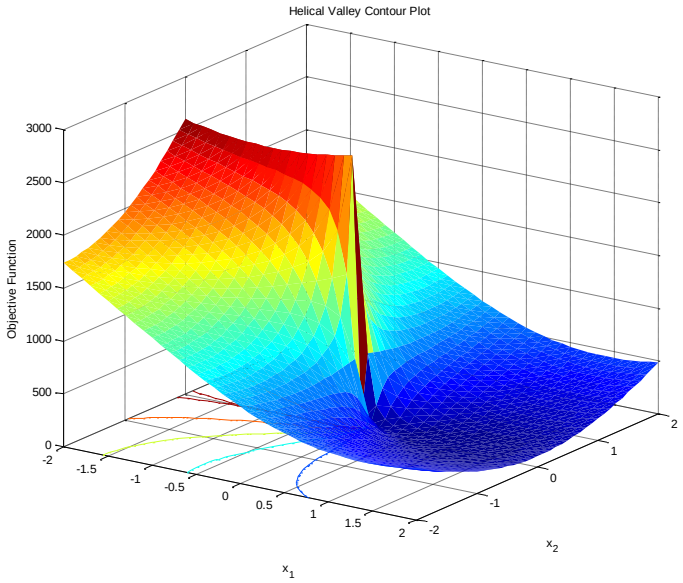


Figure 44: Surface plot for the helical valley function with $x_3 = 0$

4.6.1 Helical Valley Test Problem

The helical valley objective function is defined in Equation 14 [30].

$$f(x_1, x_2, x_3) = 100 \times \left\{ (x_3 - 10\theta(x_1, x_2))^2 + \left(\sqrt{x_1^2 + x_2^2} - 1 \right)^2 \right\} + x_3^2$$

$$2\pi\theta(x_1, x_2) = \begin{cases} arctan(x_2/x_1), & x_1 > 0 \\ \pi + arctan(x_2/x_1), & x_1 < 0 \end{cases} \tag{14}$$

$$\text{subject to } -10 \leq x_i \leq 10$$

For the helical valley problem, the global minimum is located at $x = (1, 0, 0)$ and has a function evaluation of 0. Figure 44 shows a surface plot of the function with x_1 and x_2 being varied and x_3 held constant at a value of 0. Even though this figure only represents 2 dimensional data, it gives an idea of what the function looks like.

The trajectory optimization problem has a significant amount of infeasible space where a function evaluation will not return any information. This was discussed in Section 4.4. In order to simulate that with this objective function, a hard limit was put on objective function values greater than 100. In other words, if the inputs

Table 18: Pass rates for different case sets using the helical valley test function with a cutoff at 100

Cases	1000	5000	10000	20000	40000	60000
Cases Passed	4	10	17	36	94	142
Pass Rate (%)	0.40	0.20	0.17	.18	0.24	0.24

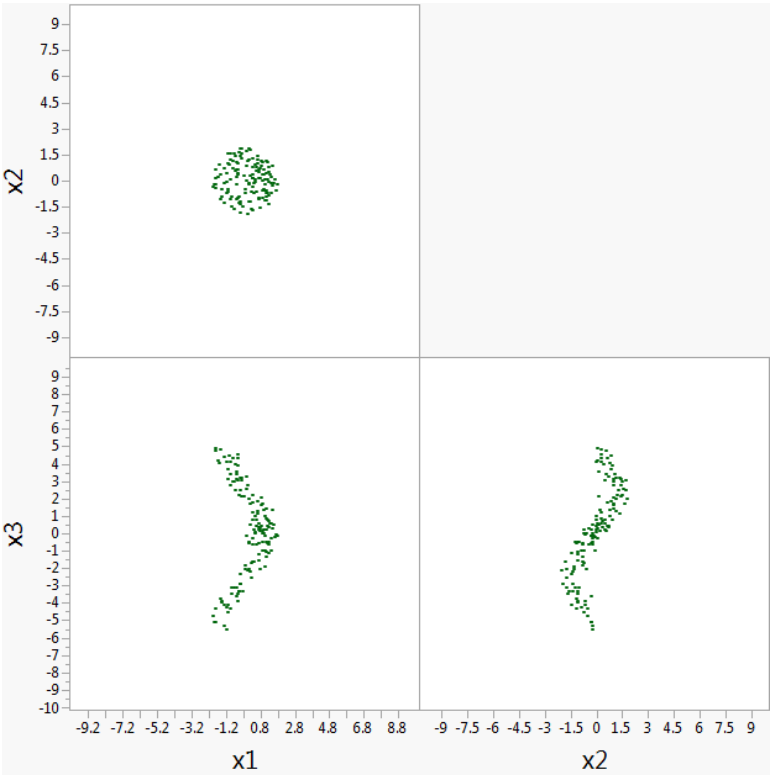


Figure 45: Feasible cases for the helical valley objective function

result in an objective function value greater than 100, the function will not return a numerical value. A set of LHC cases was run to sample the space. As discussed in Section 4.4.3.3, the pass rate can be used to determine how many cases are sufficient to adequately sample the space. From Table 18 it can be seen that around 60000 cases the pass rate is consistent. The user may choose to run more cases, but for this example 60000 cases will be used. Figure 45 shows the feasible cases from the initial 60000.

The next step in the method is to reduce the design space by using PCA to generate an uncorrelated design space, matching the distributions of the variables in

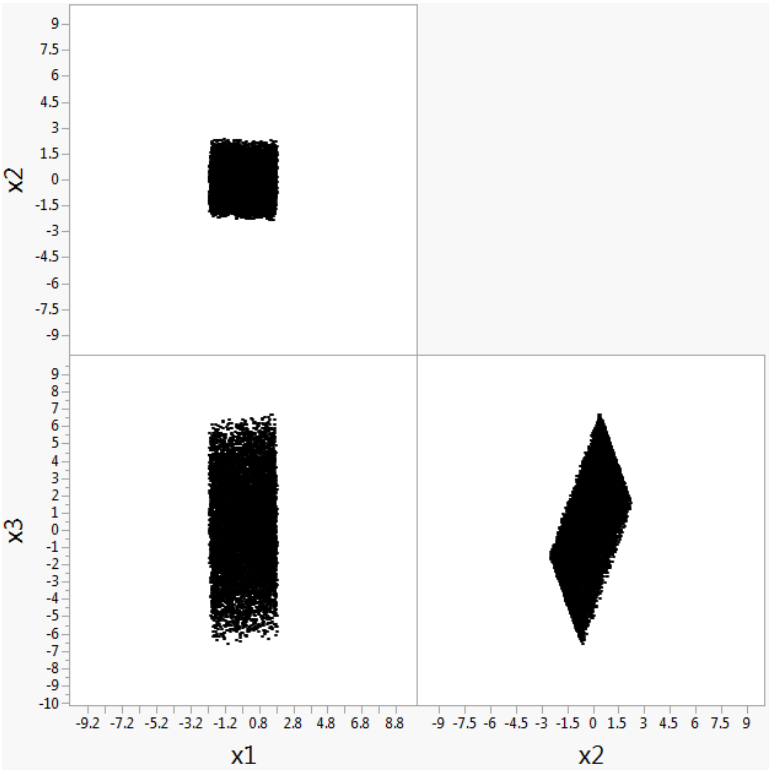


Figure 46: MC cases for the helical valley problem

the principal component space, using the distributions to generate a set of random cases, and finally mapping the cases back to the original design space. This process is explained in detail in Section 4.4.4.2. The resulting set of cases is shown in Figure 46. Overlaying the feasible data and the Monte Carlo points, seen in Figure 47, shows that the feasible space is indeed covered. If the MC cases are evaluated, 16.4% of the cases are feasible. This is a dramatic increase from the 0.24% pass rate seen in Table 18. The increase in feasible points seen in the trajectory optimization problem in Section 4.4.4.2 is greater. This is because every optimization problem is different, so the optimization algorithm will perform differently for each problem. This example, however, shows that the method presented in Section 4.4 works for the helical valley objective function.

The next step in the methodology involves local optimization using the Monte Carlo points. For the trajectory optimization problem, the local optimizer used was

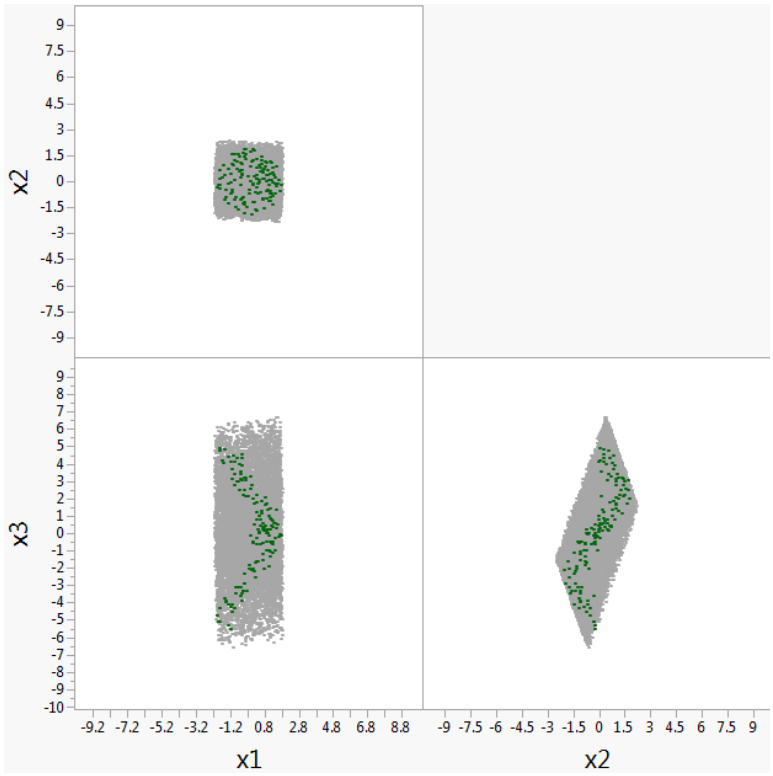


Figure 47: Overlay of feasible data and MC points for the helical valley problem

built into POST. For this problem, the MATLAB function *fmincon* [6] was used as the local optimizer. Running local optimization on the MC cases resulted in a 16.4% optimized rate. This is exactly the feasible rate for this set of data. In fact, every case that was feasible was able to be optimized. The 1639 feasible points in the MC set were all optimized to have a function evaluation of less than 2×10^{-14}. All the design variables were within 2×10^{-7} of their respective optimal values.

The helical valley function showed that the design space reduction using PCA can greatly increase the pass rate. It should be noted that this does depend on the linear correlation of the feasible points in the input space. If the points are linearly uncorrelated to start off with, this method will yield no benefit. This problem did not show that multiple local minima will be explored using the local optimizer. The local optimizer was able to find the global minima for all of the feasible cases. In order to test the local optimization part of the methodology, a second test function is used.

4.6.2 Alpine02 Test Problem

The second objective function used to test the methodology is the Alpine02 function, defined in Equation 15 [24]. In Equation 15 D is the number of dimensions. For this study, D is set to 3.

$$\begin{aligned} & f(x_1, ..., x_D) = sin(x_1)...sin(x_D)\sqrt{x_1...x_D} \\ & \text{subject to } 0 \leq x_i \leq 10 \end{aligned} \tag{15}$$

This function was originally designed as a maximization problem. Maximizing a function is equivalent to minimizing the negative of that function. Because of the way the MATLAB optimization function works, the Alpine02 problem is converted to a minimization problem by taking the negative. In this case, the global minimum is located at about $x = (7.917, 7.917, 7.917)$ and has a function evaluation of about -22.14. Figure 48 shows a surface plot of the function with x_1 and x_2 being varied and x_3 being held constant at the optimum value 7.91. Even with one of the variables being held constant, multiple local minina are clearly seen. As in Section 4.6.1, an arbitrary limit is set to define the feasible space for the Alpine02 problem. This limit is set at -10.

The feasible space can be captured with 40000 points, as seen in Table 19. The feasible points are shown in Figure 49. The methodology in Section 4.4.4.2 is carried out using this data. As a reminder, this involves generating an uncorrelated space using PCA, matching the distributions, generating a set of random cases, and mapping the cases back to the original design space. It should be noted that for this problem, the correlations are much smaller than for the helical valley. This means there will be a smaller increase in pass rate with this method. The set of MC points is shown in Figure 50. The feasible data and MC cases can be overlaid to confirm that the feasible space is covered. This is shown in Figure 51.

When these cases are run, the pass rate increases to 7.30%. This is not as large

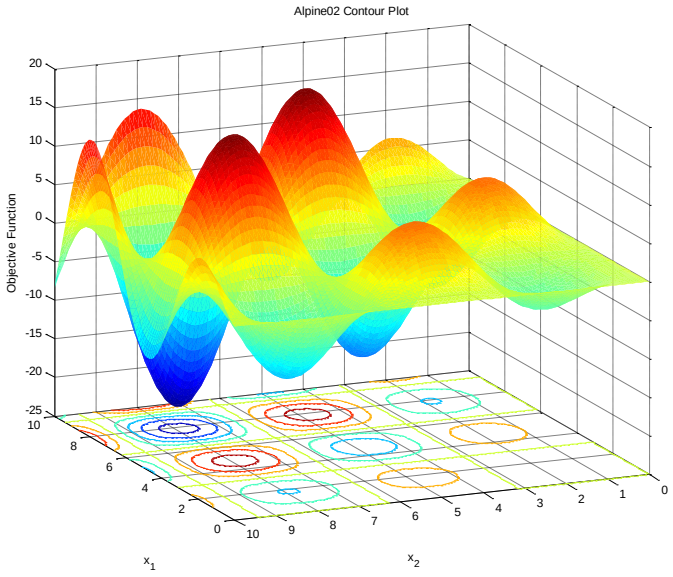


Figure 48: Surface plot for the Alpine02 function with $x_3 = 7.91$

Table 19: Pass rates for different case sets using the Alpine02 test function with a cutoff at -10

Cases	1000	5000	10000	20000	30000	40000
Cases Passed	9	59	116	231	350	466
Pass Rate (%)	0.90	1.18	1.16	1.16	1.17	1.17

of an increase in pass rate as seen in the trajectory problem or in the helical valley problem. Again, each problem is different, and from Figure 49 it can be seen that there is less correlation in the design space, which means the PCA method will not be as beneficial.

The MC cases are optimized using the MATLAB function *fmincon* [6]. The design space is multi-modal, as seen in Figure 49, so several local minima should be found. Figure 52 shows the results. As with the helical valley problem, the optimizer used does a very good job of finding local optima, so it seems there are only a few points on the plot. However, there are actually 730 points, all converging on a few local minima. The points are color-coded based on the function value, blue being the best solutions. There are a total of 9 local minima found, the best one located at $x = (7.9171, 7.9171, 7.9171)$ and with a function value of -22.144. This corresponds

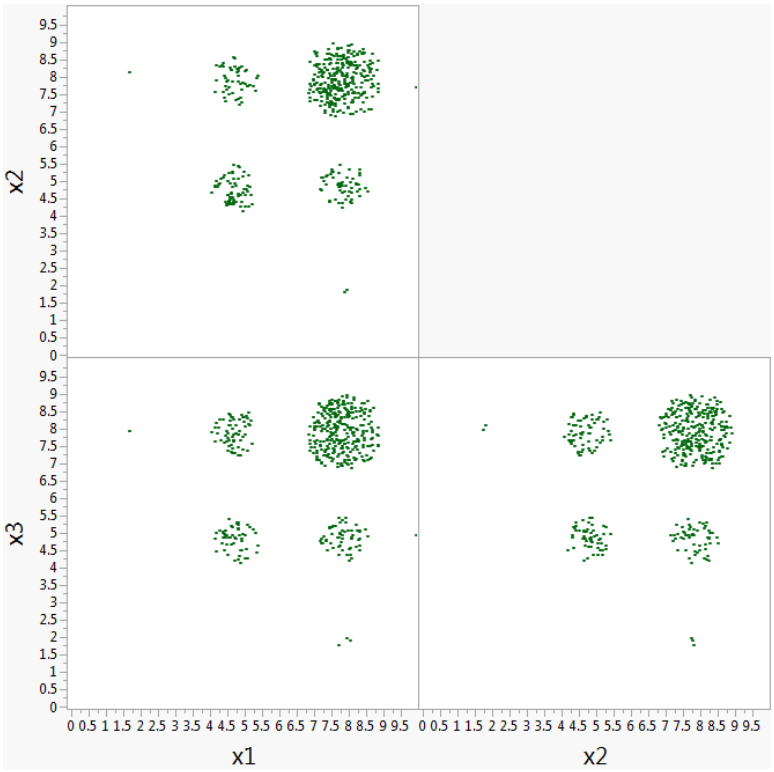


Figure 49: Feasible cases for the Alpine02 objective function

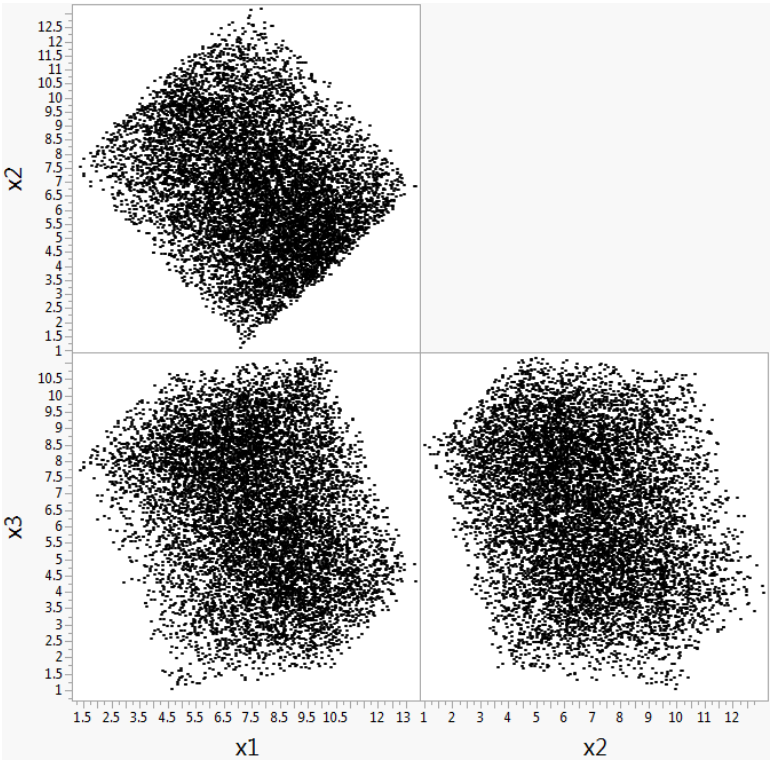


Figure 50: MC cases for the Alpine02 problem

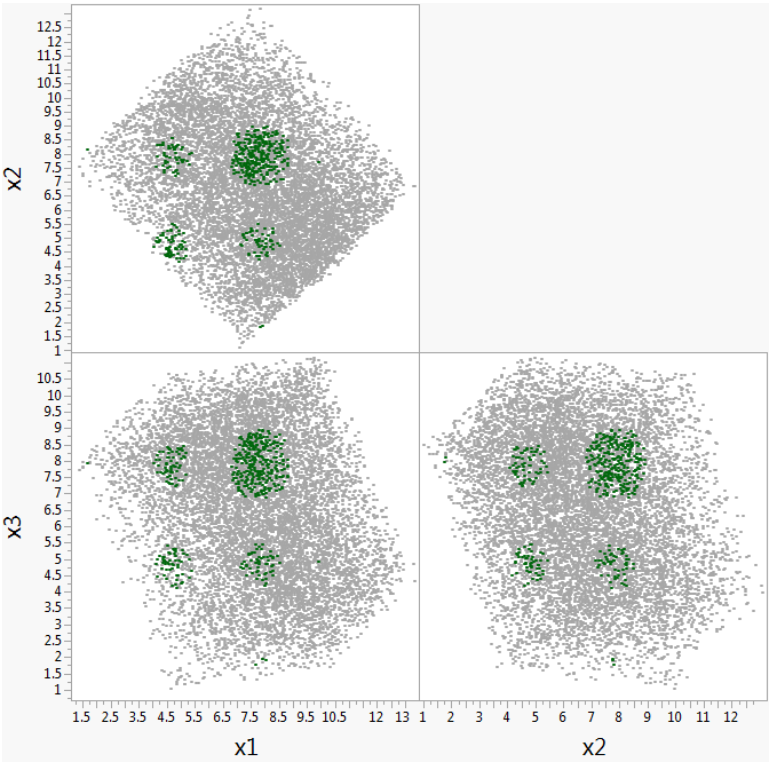


Figure 51: Overlay of feasible data and MC points for the Alpine02 problem

to the global minimum of the Alpine02 problem within the ranges considered.

The Alpine02 problem showed that the optimization method here can in fact find many local minima. In this case, 9 local minima were identified and explored, and the best of these is in fact the global minimum. It should be noted that the trajectory problem solved in this thesis is much more complicated than either of these test functions. Figure 42 shows that there is a cloud of local minima for the optimization problem, as opposed to 9 discrete points for the Alpine02 problem. However, these two test functions show that the optimization methodology presented here can be effectively applied to other optimization problems.

4.7 Experiment 5: Including Trajectory Definition in the Optimization Design Space

Up until this point, the discussion of the results on trajectory optimization has focused entirely on the control variables. Both the vehicle and the trajectory structure has

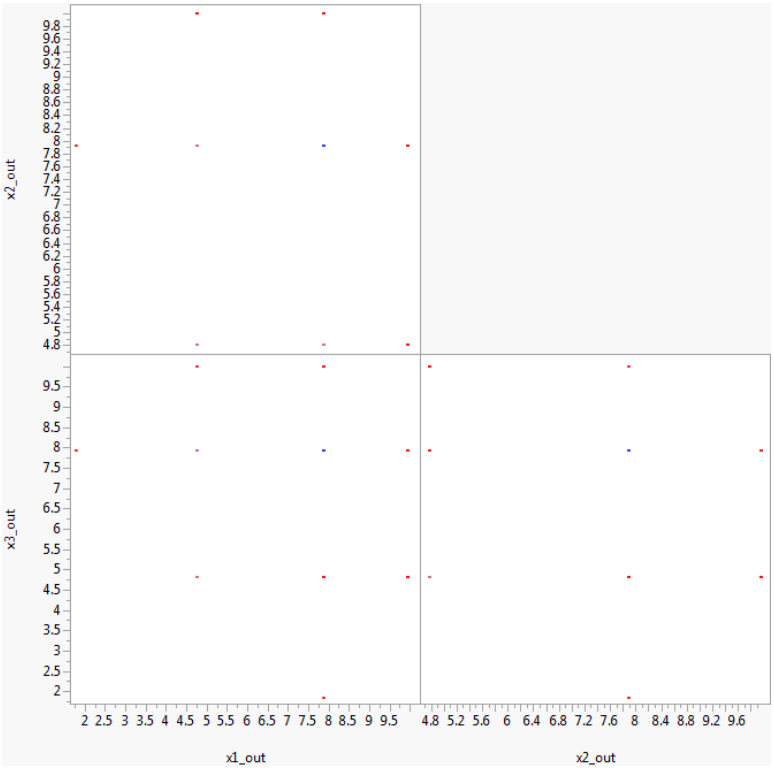


Figure 52: Optimized cases for the Alpine02 problem

remained constant. However, the trajectory structure (or phase discretization) can play an important role in the performance of the vehicle. The focus of this section is to answer Research Question 5, repeated below.

Research Question 5 - Do phase structure variables make a difference in the trajectory optimization outcome?

The discussion here on the phase discretization will be limited. The goal is to determine if phase discretization should be included in the trajectory optimization process. The goal is not to find the best strategy of including the phase variables. With that in mind, two experiments were conducted. The first kept the same number of trajectory variables and varied when the control variables apply in the trajectory.

For example, in the original trajectory formulation the control variable $u4$ applied from 200 s after staging to 500 s after staging. However, these time values were arbitrarily chosen. The control variable can be applied starting from 190 s instead of 200 s. The effects of this change were investigated. The second experiment dealt with the number of trajectory variables included in the process. For example, after staging there are 3 variables used to control the trajectory. If 6 variables are used instead, the trajectory may perform better, even though the search will take longer because of the increased dimensionality.

As stated previously, the goal is not to fully investigate the effects of adding more variables or completely understand how changing the phase structure will affect the trajectory. Instead, the goal is to determine whether enough benefit is seen from considering trajectory structure variables in the optimization process to merit their inclusion.

4.7.1 Phase Criteria Values

Keeping in mind that the objective is to determine whether or not phase criteria values should be part of the optimization process, only the phase structure for the upper stage guidance was initially varied. If better solutions resulted, no further investigation would be required. This would show that changing the trajectory structure does indeed affect the output, and should be included. If worse solutions resulted, the same can be concluded, because it would show again that changing the trajectory structure changes the output, and the current arbitrarily chosen structure happened to be a good one. If no change in the solutions resulted, further investigation would be required using more phases than just the upper stage phases.

Referring to Table 5, the upper stage controls are $u3$, $u4$, and $u5$. The beginning of the phase where $u3$ is applied is set by the first stage burnout, and cannot be changed. However, the end of that phase (which is the beginning of the phase where

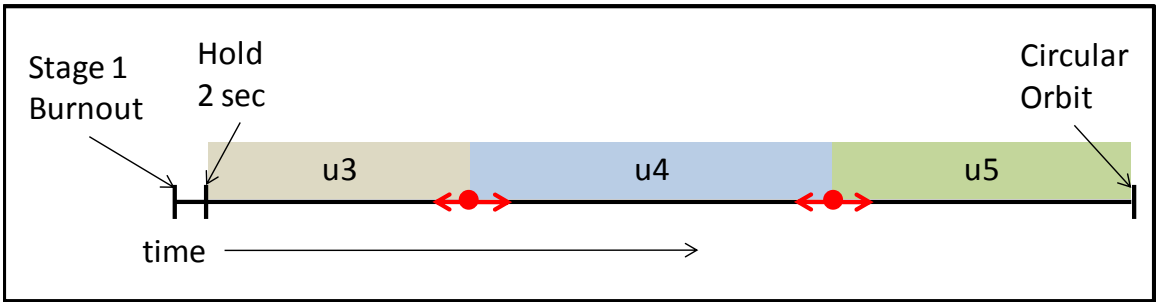


Figure 53: Timeline for upper stage trajectory structure

$u4$ is used) and the end of the next phase can be changed. Figure 53 depicts what is explained here. The red dots represent when the changes from $u3$ to $u4$ and from $u4$ to $u5$ are applied. Those times can change, as they were arbitrarily chosen. As those times are changed and the trajectory is optimized, different results are expected.

The experiment was set up using the method explained in Section 4.5.2. As the goal is to see if better results can be found, the method that has yielded the best results so far was employed. Because the trajectory structure is changed, it is not certain that the previous data applies to the problem. However, in the interest of scoping the problem, the method was used to provide initial guesses for the control variables.

The time values were set as random numbers between $\pm 50 sec$ of the current value. This was chosen arbitrarily, as the goal is to determine whether it has an effect, not to find the best trajectory structure.

As in Section 4.5.2, four sets of cases were run, two with 1000 cases each and two with 10000 each. For the best results, the method should be started from the the global search again and these additional time variables included in the process. However, the method is time consuming, so this alternate approach was used.

Table 20 displays the results of this experiment. A comparison of these results to the results given in Table 17 shows that they are very similar. Both the 1000 case sets perform better without varying the times, while both 10000 case sets perform better when the times are varied. It should be noted that the way the initial guesses for

the controls were selected was designed for the baseline times. When the times are varied, this method is no longer applicable. Therefore the fact that varying the times provided equivalent or even slightly better answers is an indication that varying the phase structure is important.

Table 20: Results of experimental sets created using the best performing cases after optimization and varying phase criteria for upper stage guidance

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	437	43.7	-1885.4
2	1000	387	38.7	-1884.1
3	10000	4147	41.5	-1815.1
4	10000	4173	41.7	-1758.7

Because this result is not very persuasive, one more experiment was performed, this time varying the phase criteria for the other controls as well. Again, because of the way the cases were selected, it is expected that the optimized rate will decrease. Even still performance may increase because of the increased freedom the optimizer has over the variables. Referring to Table 5, altitude at which the gravity turn is initialized in Phase 2 and the dynamic pressure at which Phase 2 ends were varied within $\pm 500\ ft$ and $\pm 15\ psf$ respectively. Like the range on times, these values were arbitrarily chosen to see if they had an effect. The changes to these values were added to the same experiments run previously in this section.

Table 21 shows the results of these sets. Again the results are inconclusive. Better results are seen for some of the cases, while worse for others.

Table 21: Results of experimental sets created using the best performing cases after optimization and varying all available phase criteria

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	418	41.8	-1803.9
2	1000	422	42.2	-1847.5
3	10000	4266	42.7	-1837.6
4	10000	4240	42.4	-1795.6

A final way to see if the phase structure makes a difference in how the trajectories

perform is to take a single trajectory that works well, and change the phase structure for this trajectory while keeping the initial guesses constant. This does not address the question of how to include phase structure variables in the optimization process, but it will determine if it should be included.

The best solution from this set of cases was a trajectory that ended with -1705 *lbs* of propellant remaining. It is interesting to note, however, that over 10% of the cases performed better than best result from the case sets shown in Table 21. This is expected, as the starting guess for this case set already performed very well.

The results show, then, that yes, changing the phase structure does have an effect on the trajectory result. However, the magnitude of that effect is on the order of 10 *lbs*, so it may not be worth the extra effort to include these variables in the optimization design space. A final note before the discussion on this subject is finalized is that the phase structure variables were not varied until the end of the optimization process. Including them in the beginning along with the control variables may yield different results. This effort, however, is left for a future study.

4.7.2 Number of Control Variables

The second part of investigating the role the phase structure plays in the trajectory optimization process involves the number of control variables. The test for this was quite simple. Originally, there were 5 control variables. The previous sections have laid out a methodology to obtain good initial guesses for those control variables. Each of those control variables is applied for a certain amount of time. The control variable is applicable between the phase starting and ending criteria, as seen in Table 5.

Keeping these criteria the same, each control variable is replaced by 2 control variables. For example, control variable $u3$ is applied from 2 s after staging to 202 s (see footnotes in Table 5). Now $u3$ is replaced by 2 control variables, for example $u3\_1$ and $u3\_2$. The first applies from 2 s after staging to say 102 s after staging, and

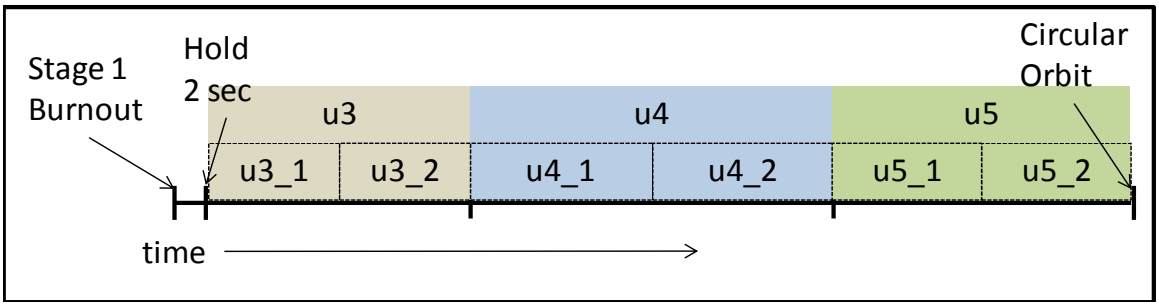


Figure 54: Timeline for upper stage trajectory structure with added control variables

the second applies from 102 s to 202 s after staging. The value that separates $u3\_1$ and $u3\_2$ is not important at this point, and was chosen to be halfway in between with respect to time.

The value of this strategy is that a method already exists to provide an initial guess for $u3$. The control variable $u3$ is applied for the same section of the trajectory as $u3\_1$ and $u3\_2$. Therefore the initial guess for $u3$ can be repeated, and used as an initial guess for both $u3\_1$ and $u3\_2$. Figure 54 depicts how this would play out for the upper stage control variables.

Initially, this was only done with the upper stage control variables to see if a difference was noticed. As in the Section 4.7.1, the goal is not to find the best way to include more variables, or the best number of variables to include, but to determine if it is worth investigating the number of variables used. If no change is seen, the method will be applied to the first stage control variables as well. Four sets of cases were used, two of 1000 cases and two with 10000. The initial guesses for the control variables were the same as the ones used in Section 4.7.1. Since the cases are selected randomly, there is no reason to select different guesses.

Table 22 shows these results. Comparing these results with the results in Table 17 shows that for 3 of the 4 sets run, using a higher number of controls improves the result. In addition, for 3 of the 4 sets the optimized rate was higher. While this is not of direct importance, a higher optimized rate will increase the probability of finding better solutions, which is the objective of these simulations. The higher optimized

rate is due to the greater flexibility the optimizer has to modify the trajectory due to the higher number of variables. Therefore, POST has more opportunity to find a trajectory that meets the constraints.

Table 22: Results from case sets using additional repeated control variables for upper stage guidance

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	450	45.0	-1861.9
2	1000	443	44.3	-1853.5
3	10000	4176	41.8	-1809.0
4	10000	4318	43.2	-1805.6

In the cases presented in Table 22, the same guess was used for both the the control variables used to replace the previous control variable. For example, the same guess was used for $u3\_1$ and $u3\_2$. However, different guesses that are still consistent with the methodology presented in Section 4.4.4.2 can be applied to $u3\_1$ and $u3\_2$. After all, these initial guesses come from a distribution, so instead of choosing a single number, two were chosen.

There are two considerations to take into account when selecting cases in this manner. The first is the possibility of getting better results. It is essentially exploring parts of the design space that have not been explored. These parts of the design space were not even available for exploration without the extra variables. The second consideration is because the number variables has changed, the design space has changed. The information gathered earlier does not necessarily apply. The correlations between $u4$ and $u5$, for example, may only be seen now between $u4\_2$ and $u5\_1$, and a completely different correlation may exist between $u4\_1$ and $u5\_2$. Therefore, the best option at this point is to start over with the global search and design space reduction. This was not done in the interest of scoping the problem. Determining the best way to add variables to the design space, or choose the best number of variables, is outside the scope of this discussion. What is dealt with here, is whether or not the number of variables should be considered.

Table 23 shows the results of these cases. The performance was actually worse than the original results in Table 17, where the phase structure was kept constant. This could be for one of two reasons. The first is that adding control variables does not help the trajectory. The second is that the design space investigation done with 5 variables, which is used here, does not apply to a design with 8 variables.

Table 23: Results from case sets using additional but different control variables for upper stage guidance

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	453	45.3	-1880.4
2	1000	481	48.1	-1865.8
3	10000	4670	46.7	-1864.9
4	10000	4612	46.1	-1862.8

A set of cases was run to try to obtain some definitive results. As in Section 4.7.1, the best case found so far was chosen. The trajectory definition was modified to have 3 more control variables, for a total of 8. Then small perturbations were run around this case to see if any improvement was found. A total of $20,000$ cases were run. These cases took significantly longer, as the local optimizer is controlling more variables.

Surprisingly, these did not find a better solution. The best case ended the trajectory with $-1743.1\,lbs$, about $27\,lbs$ lighter than the case all these trajectories were based on. This is completely opposite to what was expected. Further investigation was performed to determine what was happening. The first step was to make sure the baseline best case was working well with the new trajectory structure. This was confirmed by running the best case final control parameter values with the new trajectory structure without optimization. The outputs matched, so the trajectory setup for the new phase structure was not a problem. Running that same trajectory setup, however, with optimization yielded a worse result. This seems to go against common sense. However, the trajectory design space is extremely non-linear. Local optimizers may take a step in a direction that leads to a worse local optimum. If the first step

was taken in a different direction, the result may be drastically different. However, for this study there was no control over the local optimization algorithm details.

With this in mind, a different local optimization scheme was employed for this specific case. POST has several optimization methods available, as discussed in Section 2.2.3.1. The method used up to this point in the study was the accelerated projected gradient (PGA) method. This was because of its speed. However, to see if the optimization method made a difference, the best case was run using the projected gradient method (instead of the accelerated PGA). The number of phases was varied as well. The results, shown in Table 24, are indeed interesting.

Table 24: Comparison of different optimization methods and number of phases using current best case

Test	Number of Variables	Optimization Method	Solution (lbs)
1	5	Accelerated PGA	-1718.5
2	6	Accelerated PGA	-1901.3
3	5	PGA	-1898.3
4	6	PGA	-511.0

What is seen very clearly is that the number of variables along with the optimization method makes a tremendous difference. An improvement of over 1200 (lbs) to orbit is seen. Recall that getting a lbs of payload to orbit can cost anywhere between \$1000 and \$10000. The improvement here translates to between 1.2 to 12 million dollars.

A paragraph is taken to discuss what can and what cannot be concluded from these results. It can be concluded that the phase structure can make a large difference for a specific trajectory. Consider Test 3 and 4 in Table 24. The only difference is a degree of freedom, an added variable in the trajectory structure. The optimization method, initial guesses, and vehicle parameters were all the same. It can also be concluded that the optimization method can make a difference. This result was not sought after in this study, but it has become clear. Consider Test 2 and 4. The only difference is the optimization method. Phase structure, initial guesses, and vehicle parameters

were all the same.

What cannot be concluded from these results is that a similar or better result would not have been found if all the local optimization cases with 5 variables had been performed using the PGA optimization method. This can easily be tested by running a set of cases with the new optimization method. For direct comparison, the four sets of cases used in Section 4.5.2 were used.

Table 25 shows the results. Compared to the results in Table 17, this optimization method performs worse for these cases. The question arises, then, of what would happen if a case set were run with 6 design variables using the PGA optimization method. The results in Table 24 indicate that better results would be obtained.

Table 25: Results from cases shown in Table 17 with different optimization method

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	680	68.0	-1879.3
2	1000	720	72.0	-1888
3	10000	7026	70.3	-1835.9
4	10000	6992	69.9	-1859.1

The results are dramatically different for the same set of cases with 6 design variables instead of 5. Recall that a set of cases with 8 design variables was run. These cases added phases to the upper stage portion of the trajectory. However, after seeing that only small differences were obtained, a different phase modification was implemented. An additional phase was added to the pitch over maneuver used to initiate the gravity turn. This corresponds to Phase 2 in Table 5. So the pitch over maneuver was modified to have 2 control variables, while the entire pitch over maneuver was kept the same. The results are shown in Table 26. Comparing Table 25 and Table 26, where only the number of phases used to define the trajectory is different, shows that the number of phases can have a very large effect on the trajectory performance.

As another test, the set of $20,000$ cases around the best found solution with 8

Table 26: Results using 6 design variables and PGA optimization method

Case Set	Cases	Optimized Cases	Optimized Rate (%)	Best Solution (lbs)
1	1000	616	61.6	-482.3
2	1000	644	64.4	-482.3
3	10000	6292	62.9	-481.5
4	10000	6285	62.8	-481.7

control variables run earlier was repeated using the PGA optimization method. The results were somewhat worse, the best case yielding a final propellant of 1989 (lbs). This shows that adding phases my not always be beneficial.

4.7.3 Phase Discretization Summary

The result of the phase discretization experiment is that phase variables can indeed make a dramatic difference. However, this difference is not always seen, and may depend on other variables not included in this study, such as local optimization method. What is clearly seen, though, is that adding a control variable can make a dramatic difference. In addition, allowing the boundaries between phases to vary can also improve the result, although not as dramatically.

CHAPTER V

CONCLUSIONS

5.1 Answers to Research Questions

In this section the research questions will be repeated and summary answers given. For more information on any of the results, refer to Chapter 4.

Research Question 1 - Can an existing tool be used to evaluate trajectories for a global search?

Answer - Yes, the industry standard trajectory tool POST can be used.

Research Question 2 - Should path constraints be evaluated during the global search for all the trajectories or at the end of the global search for trajectories that meet the required ending conditions?

Answer - Using POST, path constraints should be evaluated during the global search.

Research Question 3a - How can the global search be initialized?

Answer - Between LHC and FF designs, the global search should be initialized using a LHC design.

Research Question 3b - What is an effective strategy to reduce the design space?

Answer - PCA on the feasible points can be used to generate a design space where a much larger percentage of points are feasible (compared to the initial percentage).

Research Question 4 - Will using local optimization on the results from the global search yield better solutions?
Answer - Yes, using a local optimizer to fine-tune solutions yields much better results.

Research Question 5 - Do phase structure variables make a difference in the trajectory optimization outcome?
Answer - Yes, phase structure variables can drastically affect the vehicle performance.

5.2 Contributions

There are several contributions attributed to this work.

1. Demonstrated that POST can be used as a trajectory propagator for global searches.

2. Created a method to capture the feasible space of a launch vehicle ascent trajectory problem.

3. Created a method to inform a local search based on feasible design space from the global search or "top-performing" space from the local search.

4. Successfully applied these methods to other generic optimization problems.

5. Demonstrated that trajectory phase structure variables can have a drastic effect on the trajectory optimization result.

5.3 Future Work

There are several topics dealt with in this thesis that require more research. The first is investigating the global search. Only two initialization designs were considered. In addition, the design space reduction strategy was carried out using the simplest tools available. Investigating other options for initialization, distribution fitting, and case selection would be of value.

The second area is the question of local optimization. For this thesis, a single local optimization method was used until the final experiment. The original goal was to keep the the local optimization constant and not include it as a variable. Using different local methods may work better for trajectory problems, especially if the input cases are assumed to be already feasible.

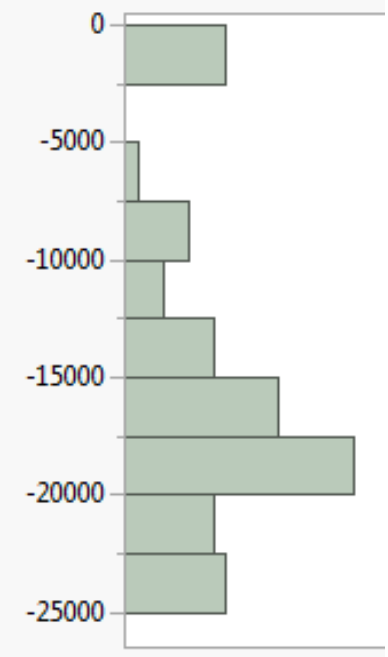
A third area is determining how phase discretization variables should be included. The work here only showed that it should be included, but did not develop a strategy for including it.

Finally, research into developing a tool made specifically to determine trajectory feasibility would speed up the global search and possibly allow for either much larger searches or much shorter run-times.

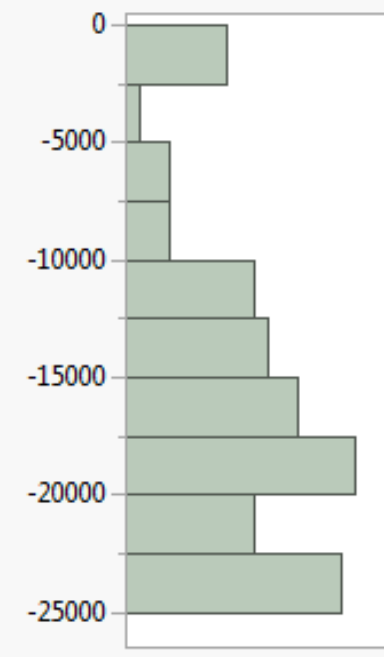
APPENDIX A

ADDITIONAL DATA AND VISUALIZATIONS

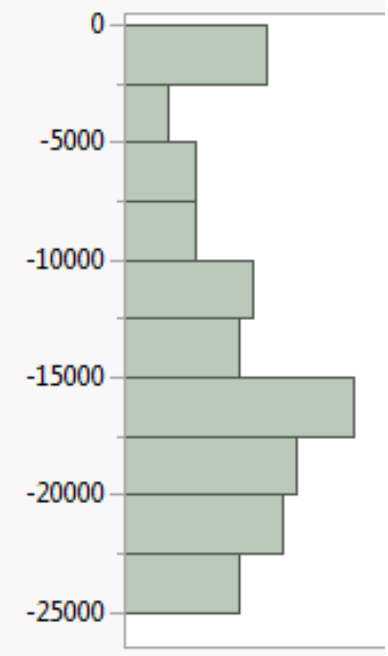
The following histograms are from the 10 repetitions of 632 case experiments in Section 4.5.1. The histograms are of propellant remaining at the end of the optimized trajectory. The repeated experiments were set up using the design space created from the global search.



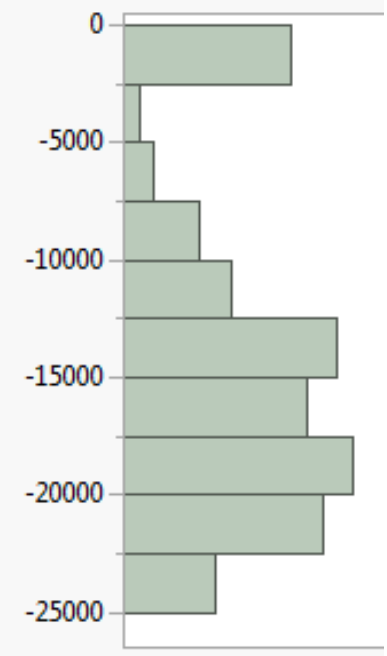
Repetition 1



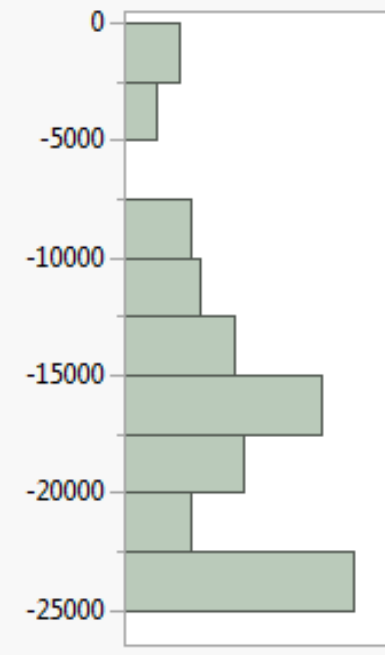
Repetition 2



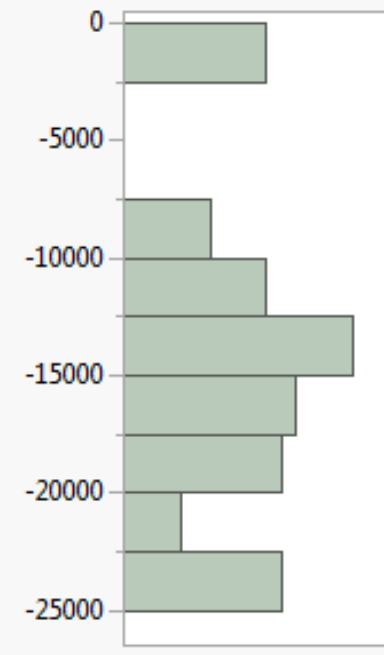
Repetition 3



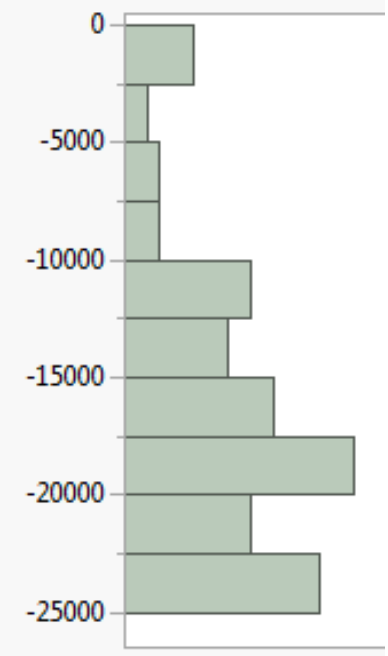
Repetition 4



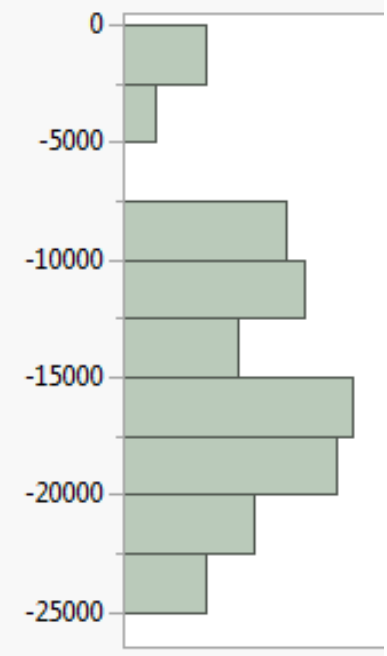
Repetition 5



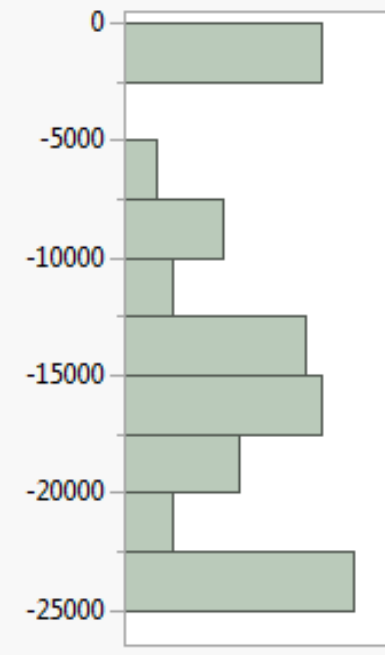
Repetition 6



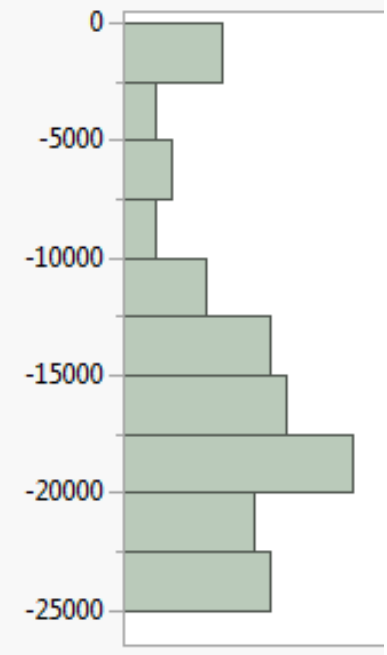
Repetition 7



Repetition 8



Repetition 9

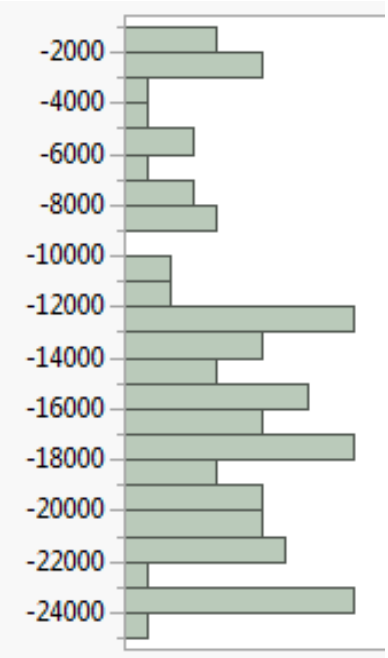


Repetition 10

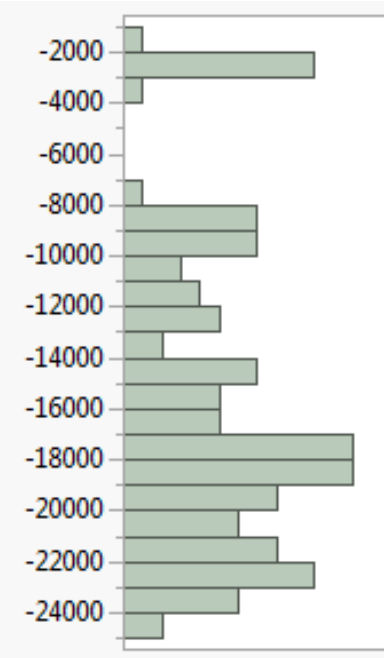
Table 27: Passed cases for repetitions of 632 case experiments

Case Set	Number of Cases	Cases Passed	Pass %
1	632	69	10.9
2	632	85	13.4
3	632	87	13.8
4	632	86	13.6
5	632	87	13.8
6	632	80	12.7
7	632	95	15.0
8	632	75	11.9
9	632	70	11.1
10	632	68	10.8
Average	632	80.2	12.7

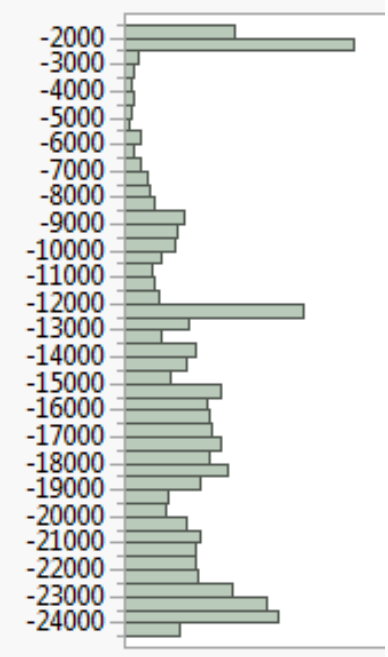
The following histograms are from the 2 repetitions of 1000 case experiments and the 2 repetitions of 10000 case experiments in Section 4.5.1. The histograms are of propellant remaining at the end of the optimized trajectory. The repeated experiments were set up using the design space created from the global search.



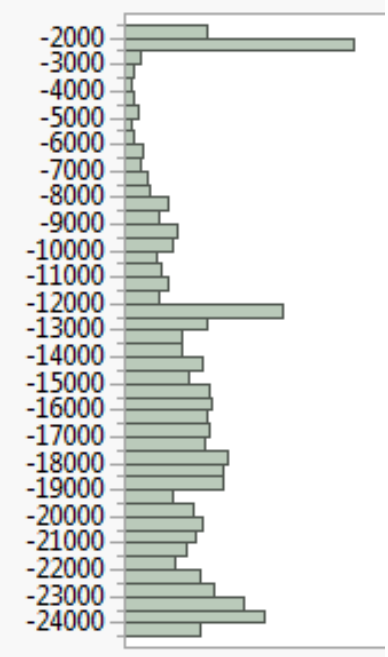
Repetition 1



Repetition 2



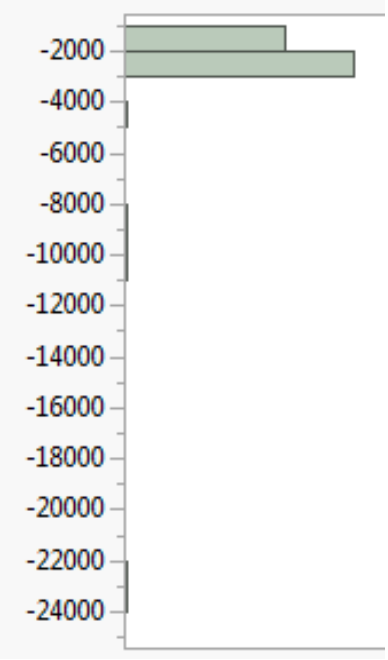
Repetition 1



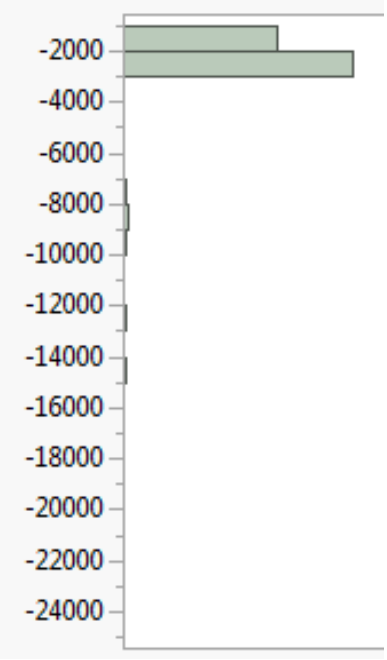
Repetition 2

The following histograms are from the 2 repetitions of 1000 case experiments, the 2 repetitions of 10000 case experiments, and the 2 repetitions of 100000 case

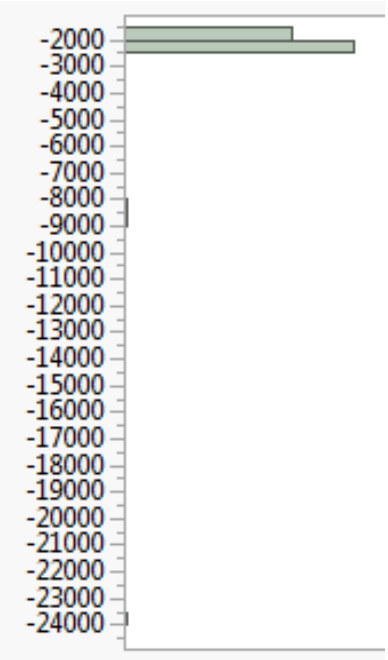
experiments in Section 4.5.2. The histograms are of propellant remaining at the end of the optimized trajectory. The repeated experiments were set up using the design space created using the top performing optimized cases.



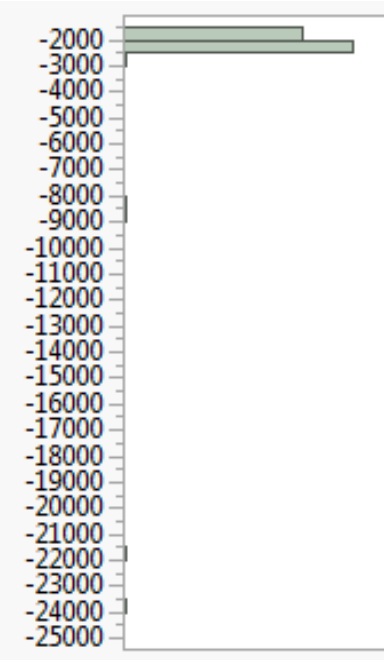
Repetition 1



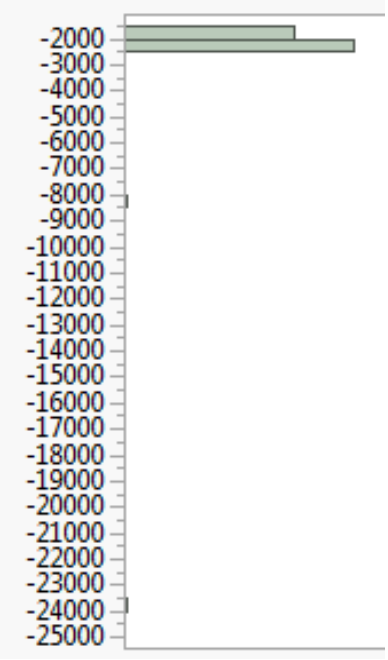
Repetition 2



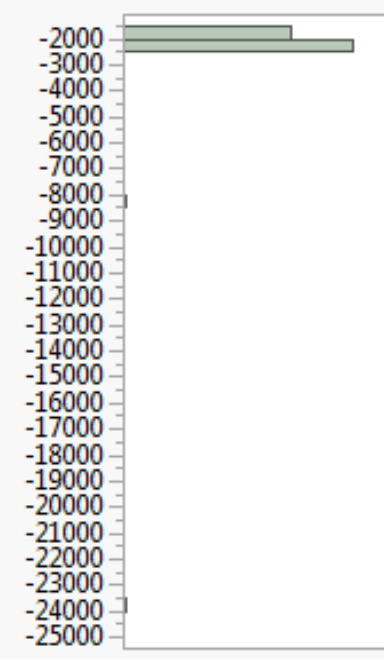
Repetition 1



Repetition 2



Repetition 1



Repetition 2

APPENDIX B

SOFTWARE

B.1 Sample POST Input File

```
C-------------------------------------------------------------
 $SEARCH
 listin   = -1,
 srchm    = 5,
 optvar   = 'WEIGHT',
 opt      = 1,
 optph    = 100,
 maxitr   = 100,
 coneps   = 91,
 wopt     = 0.00001,
 nindv    = 5,
 indvr    =     6HPITPC2, 6HPITPC2, 6HPITPC2, 6HPITPC2,6HPITPC2,
 indph    =      5, 20, 35,40,45,
 u        =  0.099634,
 0.02025,
 -1.5004,
 -1.4001,
 0.017714,
 ndepv    = 2,
 depvr    =  5HGDALT, 6HGAMMAI,
 depval   = 1312340,0,
 depph    =       100,100,
 idepvr   =         0,          0,
 deptl    =       100,        1,
 $
```

```
26      $GENDAT
27       event    = 1,
28       maxtim   = 2000,
29       altmax   = 100000000,
30       altmin   = -1000,
31       prnc     = 0,
32       prnca    = 0,
33       fesn     = 100,
34       dt       = 1,
35       pinc     = 1000,
36       time     = 0,
37      C Vehicle Weight Configuration
38       nstpl    = 1,
39       nstph    = 4,
40       wstpd(1) = 58996,
41       wstpd(2) = 6283,
42       wstpd(3) = 18500,
43       wstpd(4) = 3697,
44       wprp(1)  = 440042,
45       wprp(2)  = 44974,
46       menstp(1)= 1,
47       menstp(2)= 2,
48       mentnk(1)= 1,
49       mentnk(2)= 2,
50       sref = 219,
51      C Propulsion Parameters
52       neng     = 2,
53       nengh    = 2,
54       nengl    = 1,
55       iengmf(1)= 1,
56       iengmf(2)= 0,
57       iwdf     = 3,3,
```

```
58      iwpf     = 1,1,
59     C Guidance
60      iguid(1) = 1,
61      iguid(2) = 0,
62      iguid(4) = 1,
63      pitpc(1) = 0,
64      pitpc(2) = 0,
65     C NPC's
66      npc(5)   = 5,
67      npc(7)   = 1,
68      npc(9)   = 1,
69      npc(8)   = 2,
70      npc(19)  = 0,
71      npc(30)  = 3,
72     C Initial Conditions
73      gclat    = 28.46,
74      long     = 279.38,
75      gdalt    = 0,
76      velr     = 0,
77      azvelr   = 0,
78      gammar   = 0,
79      piti     = 0,
80      roli     = 0,
81      yawi     = 0,
82      monx(1)  = 'dynp',
83      monx(2)  = 'gdalt',
84      monx(3)  = 'gammai',
85      asmax    = 5,
86      prnt(1)  = 'time',
87      prnt(2)  = 'gdalt',
88      prnt(3)  = 'veli',
89      prnt(4)  = 'gammai',
```

```
 prnt(5)  = 'wprp2',
 prnt(6)  = 'xmax1',
 prnt(7)  = 'xmax2',
 prnt(8)  = 5HPSTOP,
 $
 $TBLMLT
 cdm(1) = 1,
 $
*include 'aero.aero'
*include 'prop_delta_iv.prop'
 $GENDAT
 event(1) = 100,
 event(2) = 1,
 critr     = 'veli',
 value     = 25548.8,
 endphs   = 1,
 $
 $GENDAT
 event    = 5,
 critr    = 'gdalt',
 value    = 3500,
 iguid(4) = 0,
 pitpc(2) = 0.099634,
 dtimr(3) = 1,
 timrf(3) = 0,
 endphs   = 1,
 $
 $GENDAT
 event    = 10,
 critr    = 'dynp',
 value    = 150,
 iguid(1) = 0,
```

```
122       iguid(2) = 1,
123       iguid(6) = 3,
124       iguid(7) = 0,
125       iguid(8) = 0,
126       desn     = 15,
127       dalpha   = 0,
128       betpc(2) = 0,
129       bnkpc(2) = 0,
130       dtimr(1) = 1,
131       timrf(1) = 0,
132       endphs   = 1,
133       $
134       $GENDAT
135       event    = 15,
136       critr    = 'timrf1',
137       value    = 10,
138       iguid(1) = 0,
139       iguid(2) = 0,
140       iguid(4) = 0,
141       alppc(2) = 0,
142       betpc(2) = 0,
143       bnkpc(2) = 0,
144       endphs   = 1,
145       $
146       $GENDAT
147       event(1) = 26,
148       event(2) = 1,
149       critr    = 'wprp1',
150       value    = 0,
151       mdl      = 3,
152       nstpl    = 2,
153       nengl    = 2,
```

```
154       npc(8)    = 0,
155       dtimr(2)  = 1,
156       timrf(2)  = 0,
157       endphs    = 1,
158       $
159       $GENDAT
160       event     = 20,
161       critr     = 'dynp',
162       value     = 20,
163       mdl       = 3,
164       iguid(1)  = 1,
165       iguid(2)  = 0,
166       iguid(4)  = 0,
167       pitpc(2)  = 0.02025,
168       endphs    = 1,
169       $
170       $GENDAT
171       event     = 35,
172       critr     = 'timrf2',
173       value     = 2,
174       mdl       = 1,
175       iengmf(2)= 1,
176       timrf(2)  = 0,
177       pitpc(2)  = -1.5004,
178       endphs    = 1,
179       $
180       $GENDAT
181       event(1)  = 37,
182       critr     = 'timrf2',
183       value     = 10,
184       nstph     = 3,
185       endphs    = 1,
```

```
186       $
187       $GENDAT
188       event     = 40,
189       critr     = 'timrf2',
190       value     = 200,
191       pitpc(2) = -1.4001,
192       endphs    = 1,
193       $
194       $GENDAT
195       event     = 45,
196       critr     = 'timrf2',
197       value     = 500,
198       pitpc(2) = 0.017714,
199       endphs    = 1,
200       $
201       $GENDAT
202       event          = 400,
203       critr          = 'times',
204       value          = 200000,
205       endphs         = 1,
206       endprb         = 1,
207       endjob         = 1,
208       $
```

B.2 MATLAB Code to Perform PCA

```
clear;clc;
%Import data
filename = 'data.txt';
data = dlmread(filename);
var_num = size(data,2);
means = mean(data);
standard_dev = std(data);
[coeff,score,latent,tsquare] = princomp(zscore(data));
offset = means*coeff;
hold = (data-repmat(means,size(data,1),1))./repmat(standard_dev,size
    (data,1),1)*coeff;
test = (score*coeff').*repmat(standard_dev,size(data,1),1) + repmat(
    means,size(data,1),1);
mc_size = 10000;
mc = lhsdesign(mc_size,var_num).*repmat(range(score),mc_size,1)+
    repmat(min(score),mc_size,1);
% Transform mc back into original design space variables
mc_transform = mc*coeff'.*repmat(standard_dev,mc_size,1) + repmat(
    means,mc_size,1);
range_mins = [-0.3177,-2,-2,-2,-1.2563];
range_maxs = [0.3254,2,2,2,1.2463];
mc_transform_scale = (mc_transform-repmat(min(mc_transform),mc_size
    ,1))./repmat(range(mc_transform),mc_size,1);
mc_transform_rescale = mc_transform_scale.*repmat(range_maxs-
    range_mins,mc_size,1)+repmat(range_mins,mc_size,1);

%The second part of this code fits distrubutions on the data
    obtained and uses those distributions to get the MC
mins = min(score)+repmat(.000001,1,var_num);
maxs = range(score)+repmat(.0001,1,var_num);
```

```
24      scaled_score = (score-repmat(mins,size(score,1),1))./repmat(maxs,
            size(score,1),1);
25      mc_v2 = zeros(mc_size,var_num);
26      for i = 1:var_num
27          [dist, pd] = allfitdist(scaled_score(:,i));
28          mc_v2(:,i) = random(pd{1},mc_size,1).*repmat(maxs(i),mc_size,1)
                + repmat(mins(i),mc_size,1);
29      end
30      mc_v2_transform = mc_v2*coeff'.*repmat(standard_dev,mc_size,1) +
            repmat(means,mc_size,1);
31      %Write data to file
32      name = 'transformed_set';
33      fid = fopen(name,'w');
34      dlmwrite(name,mc_v2_transform)
35      fclose('all')
```

REFERENCES

[1] "U.S. Space Transportation Policy," 2005.

[2] "2012 Commercial Space Transportation Forecasts," 2012.

[3] "FY 2012 Year in Review," tech. rep., Federal Aviation Administration, 2012.

[4] "Delta iv launch services user's guide," tech. rep., United Launch Alliance, June 2013.

[5] "Matlab statistics toolbox," tech. rep., The MathWorks, Inc., Natick, MA, 2013.

[6] "Matlab optimization toolbox," tech. rep., The MathWorks, Inc., Natick, MA, 2014.

[7] A. E. Bryson, J. and Ho, Y. C., *Applied Optimal Control*. New York: Wiley, 1975.

[8] Addis, B., Cassioli, A., Locatelli, M., and Schoen, F., "Global Optimization for the Design of Space Trajectories."

[9] Anisi, D., "On-Line Trajectory Planning Using Adaptive Temporal Discretization," tech. rep., Royal Institute of Technology, Sweden, 2006.

[10] Anisi, D. A., "Adaptive Node Distribution for On-line Trajectory Planning," 2006.

[11] Arora, J. S., *Introduction to Optimum Design*. Elsevier, Inc., second edition ed., 2004.

[12] Ascher, U. M. and Petzold, L. R., *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.

[13] Atkinson, K. E., *An Introduction to Numerical Analysis*. John Wiley & Sons, Inc., 2nd ed., 1989.

[14] Bate, R. R., Mueller, D. D., and White, J. E., *Fundamentals of Astrodynamics*. Dover Publications, Inc., 1st ed., 1971.

[15] Bayley, D. J., *DESIGN OPTIMIZATION OF SPACE LAUNCH VEHICLES USING A GENETIC ALGORITHM*. PhD thesis, Auburn University, 2007.

[16] Betts, J. T., "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[17] Blake, W., Rosema, C., Doyle, J., Auman, L., and Underwood, M., *Missile DATCOM*. Air Force Research Laboratory, March 2011.

[18] Brauer, G. L., Cornick, D. E., Habeger, A., Petersen, F. M., and Stevenson, R., "Program To Optimize Simulate Trajectories (POST) Volume I: Formulation Manual," tech. rep., NASA Langley Research Center, 1975.

[19] Brauer, G. L., Cornick, D. E., and Stevenson, R., "Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)," tech. rep., Martin Marietta Corporation, 1977.

[20] Burger, M., "Infinite-dimensional Optimization and Optimal Design," 2003.

[21] Castellini, F., Lavagna, M. R., Riccardi, A., and Buskens, C., "Multidisciplinary Design Optimization Models and Algorithms for Space Launch Vehicles," in *AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*,

Series Multidisciplinary Design Optimization Models and Algorithms for Space Launch Vehicles, (Fort Worth, TX), 2010.

[22] Chenglong, H., Xin, C., and Leni, W., "Optimizing RLV Ascent Trajectory Using PSO Algorithms," in *Systems and Control in Aerospace and Astronautics, ISSCAA 2008*, Series Optimizing RLV Ascent Trajectory Using PSO Algorithms, 2008.

[23] Ching, H., Ko, P.-H., Kong, C., and Du, R., "Adaptive node placement for optimal control," ICROS-SICE International Joint Conference 2009, August 2009.

[24] Clerc, M., "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, IEEE, 1999.

[25] Conway, B. A., ed., *Spacecraft Trajectory Optimization*. Cambridge University Press, 2010.

[26] Das, S. and Suganthan, P. N., "Differential evolution: A survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, 2011.

[27] Dieter, G. E., Schmidt, L. C., and others, *Engineering design*. McGraw-Hill Higher Education, 4th ed., 2009.

[28] Eagle, C. D., "A MATLAB Script that Demostrates Aerospace Trajectory Optimization Using Direct Transcription and Collocation."

[29] Elnagar, G., Kazemi, M. A., and Razzaghi, M., "The pseudospectral legendre method for discretizing optimal control problems," *IEEE Transactions on Automatic Control*, vol. 40, no. 10, 1995.

[30] Fletcher, R. and Powell, M. J., "A rapidly convergent descent method for minimization," *The Computer Journal*, vol. 6, no. 2, pp. 163–168, 1963.

[31] Frank, J., "Collocation Methods," 2008.

[32] Garrison, W., "Creation of a Trajectory Simulation Environment for Rocket-Back Booster Systems," tech. rep., Georgia Institute of Technology, Aerospace Systems Design Lab, 2011.

[33] Geeraert, J. L., "Sensitivity Analysis of 2nd Stage Launch Vehicle Guidance Algorithms," tech. rep., Georgia Institute of Technology, Aerospace Systems Design Lab, 2012.

[34] Germain, B. S. and Feld, K., "Investigation of Reusable Booster System (RBS) Trajectories," 2011.

[35] Giri, R. and Ghose, D., "Differential Evolution Based Ascent Phase Trajectory Optimization for a Hypersonic Vehicle," in *Swarm, Evolutionary, and Memetic Computing*, Chennai, India: First International Conference on Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, 2010.

[36] Hargraves, C. R. and Paris, S. W., "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1986.

[37] Hargraves, C. R., Paris, S. W., and Vlases, W. G., "OTIS Past, Present and Future," in *AIAA*, Series OTIS Past, Present and Future, 1993.

[38] Hull, D. G., "Conversion of Optimal Control Problems into Parameter Optimization Problems," *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 1, pp. 57–60, 1997.

[39] Isakowitz, S. J., Hopkins, J. P., and Hopkins, J. B., *International reference guide to space launch systems*. American Institute of Aeronautics and Astronautics, 4th ed., 2004.

[40] Jamil, M. and Yang, X.-S., "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[41] JMP, A. and Proust, M., *Modeling and Multivariate Methods*, 2012.

[42] Jolliffe, I., *Principal Component Analysis*. Springer, second ed., 2002.

[43] Kamada, R., *Trajectory Optimization Strategies for Supercavitating Vehicles*. PhD thesis, Georgia Institute of Technology, 2005.

[44] Ko, P.-H. and Du, R., "Adaptive node placement for optimal control on system of multi-degree freedom," SICE Annual Conference 2010, August 2010.

[45] Lee, K., "Introduction to Principal Component Analysis and Its Application for Reduced-Order Modeling," tech. rep., Georgia Institute of Technology, November 10, 2010 2010.

[46] Ley, W., Wittmann, K., and Hallmann, W., *Handbook of Space Technology*. Wiley & Sons, Inc., 2009.

[47] Liberti, L., "Introduction to Global Optimization," tech. rep., LIX, ÂtEcole Polytechnique, 2008.

[48] Malanowski, K., Büskens, C., and Maurer, H., "Convergence of approximations to nonlinear optimal control problems," *Lecture Notes in Pure and Applied Mathematics*, pp. 253–284, 1997.

[49] Mathews, J. H. and Fink, K. D., *Numerical Methods Using Matlab*. Prentice-Hall, Inc., 3rd ed., 1999.

[50] Mavris, D., "Class Lectures Advanced Design Methods AE 6373," tech. rep., Georgia Institute of Technology, 2010.

[51] Murray, W., Saunders, M. A., and Gill, P. E., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," tech. rep., Systems Optimization Laboratory, Stanford University, 2008.

[52] Musk, E., "Senate Testimony May 5, 2004," May 5, 2004 2004.

[53] Nelson, D., "Qualitative and Quantitative Assesment of Optimal Tarjectories by Implicit Simulation (OTIS) and Program to Optimize Simulated Trajectories (POST)," tech. rep., Georgia Institute of Technology, 2001.

[54] Paterlini, S. and Krink, T., "Differential evolution and partical swarm optimisation in partitional clustering," *Computational Statistics and Data Analysis*, vol. 50, no. 5, pp. 1220–1247, 2006.

[55] Peter, J., Carrier, G., Bailly, D., Klotz, P., Marcelet, M., and Renac, F., "Local and Global Search Methods for Design in Aeronautics," 2011.

[56] Powell, R. W., Striepe, S. A., Desai, P. N., Braun, R. D., Brauer, G. L., Cornick, D. E., Olson, D. W., and Petersen, F. M., "Program To Optimize Simulate Trajectories (POST) Volume II: Utilization Manual," tech. rep., NASA Langley Research Center, 1997.

[57] Rao, A. V., "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[58] Rasky, D. J., Pittman, R. B., and Newfield, M. E., "The Reusable Launch Vehicle Challenge," in *AIAA Space 2006*, Series The Reusable Launch Vehicle Challenge, (San Jose, CA), 2006.

[59] Rea, J., "Launch vehicle trajectory optimization using a legendre pseudospectral method," in *Proceedings of the AIAA Guidance, Navigation and Control Conference, Paper No. AIAA*, vol. 5640, 2003.

[60] Riccardi, A., Castellini, F., Buskens, C., and Lavagna, M., "Global and Local Optimization Approaches for Launch Vehicles Ascent Trajectory Design," in *62nd International Astronautical Congress*, Series Global and Local Optimization Approaches for Launch Vehicles Ascent Trajectory Design, (Cape Town, SA), International Astronautical Federation, 2010.

[61] Riehl, J. P., Paris, S. W., and Sjauw, W. K., "Comparison of Implicit Integration Methods for Solving Aerospace Trajectory Optimization Problems," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Series Comparison of Implicit Integration Methods for Solving Aerospace Trajectory Optimization Problems, (Keystone, CO), 2006.

[62] Sheppard, M., "Fit all valid parametric probability distributions to data," tech. rep., MATLAB File Exchange, 2012.

[63] Shippey, B. M., *Trajectory Optimization using Collocation and Evolutionary Programming for Constrainted Nonlinear Dynamical Systems*. PhD thesis, University of Texas at Arlington, 2008.

[64] Smith, T. K., "Interim access to the international space station," Master's thesis, Utah State University, 2010.

[65] Stryk, O. v. and Burlisch, R., "Direct and Indirect Methods for Trajectory Optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, 1992.

[66] Suli, E., "Numerical solution of ordinary differential equations," April 2013. Lecture Notes at University of Oxford.

[67] Torn, A. and Zilinskas, A., "Global optimization," *Lecture Notes in Computer Science*, vol. 350, 1989.

[68] Vanderplaats, G. N., *Multidiscipline Design Optimization*. Colorado Springs, CO: Vanderplaats Research & Development, Inc., 2007.

[69] Vukelich, S. R., Stoy, S. L., Burns, K. A., Castillo, J. A., and Moore, M. E., *Missile DATCOM*. McDonnell Douglass Missile Systems Company, December 1988.

[70] Wackerly, D. D., III, W. M., and Scheaffer, R. L., *Mathematical Statistics with Applications*. Thomson Learning, Inc., 2002.

[71] Wilhite, A., "Class Lectures in Spacecraft Desing I AE 6322," tech. rep., Georgia Institute of Technology, 2011.

[72] Womersley, R., "Local and Global Optimization: Formulation, Methods and Applications," 2008.