The Thesis Committee for Alan Robert Campbell

Certifies that this is the approved version of the following thesis:

Numerical Analysis of Complex-Step Differentiation in Spacecraft Trajectory Optimization Problems

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: David Hull

Cesar Ocampo

Numerical Analysis of Complex-Step Differentiation in Spacecraft Trajectory Optimization Problems

by

Alan Robert Campbell, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2011

I did this for me.

Acknowledgments

I would like to thank my advisor, David Hull, and the rest of my professors for their guidance in my studies. I am especially indebted to my family, friends, and classmates over the years for providing support and supplementing my desire to achieve all that I have.

Abstract

Numerical Analysis of Complex-Step Differentiation in Spacecraft Trajectory Optimization Problems

Alan Robert Campbell, M.S.E.
The University of Texas at Austin, 2011

Supervisor: David Hull

An analysis of the use of complex-step differentiation (CSD) in optimization problems is presented. Complex-step differentiation is a numerical approximation of the derivative of a function valid for any real-valued analytic function. The primary benefit of this method is that the approximation does not depend on a difference term; therefore round-off error is reduced to the machine word-length. A suitably small choice of the perturbation length, h, then results in the virtual elimination of truncation error in the series approximation. The theoretical basis for this method is derived highlighting its merits and limitations. The Lunar Ascent Problem is used to compare CSD to traditional forward differencing in applications useful to the solution of optimization problems. Complex-step derivatives are shown to sufficiently apply in various interpolation and integration methods, and, in fact, consistently outperform traditional methods. Further, the Optimal Orbit Transfer Problem is used to

test the accuracy, robustness, and runtime of CSD in comparison to central differencing. It is shown that CSD is a considerably more accurate derivative approximation which results in an increased robustness and decreased optimization time. Also, it is shown that each approximation is computed in less time using CSD than central differences. Overall, complex-step derivatives are shown to be a fast, accurate, and easy to implement differentiation method ideally suited for most optimization problems.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work intends to provide a thorough analysis of the use of a relatively new type of numerical differentiation method, complex-step differentiation, in optimization problems. This is done, in particular, in two ways. First, the applicability of this method is tested for several tools used in the solution of optimization problems, and, second, the performance is examined in comparison to traditional methods.

To be clear, complex-step differentiation (CSD) has existed in some form since, at least, Lyness & Moler (1967)[1] presented their proof for using the Cauchy theorem to calculate numerical derivatives for analytical functions. Examples of the implementation, and simplification of the method presented by Lyness and Moler, do not appear for some time after. In its present form, Squire & Trapp (1998)[2] for the mathematics and Martins, Sturdza & Alonso (2003)[3] for the numerical implementation appear to be defining works. In addition, Lai's Ph.D dissertation (2006)[4] generalizes the approach and details implementation into a Kalman filter. Shampine (2007)[5] gives a detailed approach for a general implementation of CSD into MATLAB and examines

numerical issues with the use of complex arithmetic. Finally, Lantoine *et al.* (2009)[6] outlines an elegant extension to higher order derivatives and its implementation and Arora *et al.* (2009)[7] examines the application of CSD into parallel processing and sensitivity analysis in optimization.

These works, however, do not present a thorough comparison of CSD with traditional methods. While Lantoine has an accuracy comparison for higher-order derivatives, and Arora looks at runtimes, there does not appear to be published results of basic functionality and performance. In Chapter 3 complex-step derivatives are used and compared with forward differences to determine their applicability in 1st, 2nd, and 3rd order interpolation schemes as well as in fixed-step and variable-step integration all within the framework of the Lunar Ascent Problem. Chapter 4 applies complex-step derivatives to a more complicated problem, an optimal orbit transfer, and compares these derivatives to central differences to determine their relative accuracy, robustness, and computational cost. It is the desire to explicitly show the ability of complex-step derivatives to be used in as wide range of solution methods and to show a marked improved in performance over traditional methods. But first, Chapter 2 details several traditional methods of numerical differentiation, derives the basis for complex-step differentiation, and examines positives and negatives of both.

Chapter 2

Complex-Step Differentiation

2.1 Background

Beginning with the advent of the digital computer, numerical approximation of first-order derivatives has been an important area of study for numerous applications. Sensitivity analysis, optimization, and nonlinear algebraic equation solution represent only a fraction of the applications where Jacobian matrices, derivatives, or gradients are necessary. Many of the traditional (and simplest) methods rely on properties of the definition of a derivative[8], that is

$$f'(x) = \lim_{h \to \infty} \frac{f(x+h) - f(x)}{h}. \tag{2.1}$$

The difficulty of numerical differentiation arises in the implications of numerical representation. There is an intrinsic error in computational methods which is a result of modeling real numbers as bits of data. This is particularly evident in the case of repeating decimals or irrational numbers: no matter how many digits or bits are used to represent $1/3$ or π, these numbers cannot be used without some error. This error is commonly known as round-off error,

more strictly defined as any error resulting from a computational approximation to a real number.

Another common source of numerical error is known as truncation error, defined as the resulting error from taking a finite number of steps in computation. Since many functions (such as trigonometric or exponential functions) are implemented numerically as series expansions, truncation error results when these series are truncated to a finite order.

A subset of round-off error that is often a concern is subtractive error. This can be defined as the loss of significant figures when similar numbers are subtracted. For example, the subtraction of $1.00002 - 1.00001$ gives an answer with only a single significant digit when two numbers with six significant digits are subtracted. The limiting of significant figures often leads to large round-off error when finite word-length arithmetic is involved.

Different methods of numerical differentiation are valued according to a trade-off of accuracy, speed, and ease of implementation according to the needs of the user. In general, a particular method can excel at one or two of these considerations, but it is rare to find a method that is truly fast, accurate, and easy to implement. This is a result of numerical considerations (including error) that must be taken into account for any user of numerical differentiation. This thesis intends to highlight the advantages of one particular method, complex-step differentiation, that is relatively fast, nearly exact, and quite easy to implement for a specific, yet wide, class of problems. First, a class of traditional methods which will be used as the primary basis of comparison

are detailed.

2.2 Difference Methods of Numerical Differentiation

In this section, traditional difference methods of numerical differentiation are highlighted with particular attention paid to their advantages and disadvantages in terms of speed, accuracy and implementation. The three major forms of differencing are given with derivation and notes regarding their use. Forward, central, and backward differencing are the three primary difference operators used to approximate the first derivative. Each of these are obtained from variations on the Taylor series expansion of the definition of the derivative given in Equation 2.1. The Taylor series is an infinite series representing any infinitely differentiable function, in traditional notation given by

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n. \tag{2.2}$$

In expanded notation, the Taylor series looks like

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots . \tag{2.3}$$

A first-order truncation of the Taylor series is often used in difference methods, however, higher order formulas do exist and are used. Higher order difference methods do require several additional function evaluations resulting in a trade-off between the added accuracy and the additional computational cost.

2.2.1 Forward Differencing

Arguably the most common method of numerical differentiation is the forward difference. This method is derived from the Taylor series expansion of a positive perturbation (h) of the function $f(x)$, that is,

$$f(x+h) = f(x) + hf'(x) + h^2\frac{f''(x)}{2} + h^3\frac{f''(x)}{3!} + \cdots . \tag{2.4}$$

Dropping terms of the order h^2 and above and solving for $f'(x)$ gives

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{2.5}$$

a first-order approximation for the first derivative of a function.

Implementation of the forward difference method is quite simple, merely calculate the value of the function of interest at a point, x, and a nearby point, $x+h$, subtract and divide by the value of the perturbation. There are, however, several concerns that must be addressed for best performance of this method.

Firstly, the choice of the perturbation length, h, is usually arbitrary, but certainly not trivial. Analytically, as can be seen in Equation 2.1, choice of an infinitely small h leads to the exact value of the derivative. Clearly, this is not a suitable choice numerically because there are only a finite number of digits that can be represented in the computer memory. In addition, there is the concern of subtractive error. It would seem that a very small value of h would be a good choice because it would minimize truncation error, but these

values increase the presence of round-off error. Conversely, relatively large values of h do not suffer as much from the effect of round-off error but are increasingly affected by truncation error. There exists a trade-off point, which is primarily problem dependent, that minimizes the total error by balancing the effects of truncation and round-off error.

A suitable choice for h here is

$$h = \epsilon(1 + |x|), \tag{2.6}$$

where ϵ, as a rule of thumb, is 1.0×10^{-8} for forward differencing unless otherwise determined empirically. Another recommendation[9] for the choice of h is

$$h = \sqrt{\kappa}x \tag{2.7}$$

where κ represents the machine precision (on the order of 1.1×10^{-16} for 64-bit binary machines).

The existence of truncation and round-off error, along with the necessary presence of subtraction error leaves something to be desired in terms of the accuracy of forward differences. While a suitable choice of h results in forward differencing being accurate enough for most problems, particularly sensitive or poorly scaled problems may often suffer or fail with this method. For relatively simple problems, the ease of implementation and relative speed make forward differencing a widely popular method.

2.2.2 Backward Differencing

Closely related to the forward difference, the backward difference is derived from the expansion of the Taylor series about a negative perturbation (h) of the function $f(x)$, that is,

$$f(x-h) = f(x) - hf'(x) + h^2\frac{f''(x)}{2} - h^3\frac{f^{(3)}(x)}{3!} + \cdots . \tag{2.8}$$

Dropping terms of the order h^2 and above and solving for $f'(x)$ gives

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}, \tag{2.9}$$

for the first-order approximation of $f'(x)$. Backward differencing has all of the same advantages and disadvantages of forward differencing as given in Section 2.2.1. The only difference is the direction of the perturbation step used to calculate the derivative approximation.

2.2.3 Central Differencing

If forward differencing is the most common method of numerical differentiation, central differencing is most likely a close second. The derivation of this formula is built on both the forward and backward difference derivation.

Subtracting Equation 2.8 from Equation 2.4 gives

$$f(x+h)-f(x-h) = f(x)-f(x)+hf'(x)+hf'(x)+h^2\frac{f''(x)}{2}-h^2\frac{f''(x)}{2}+\cdots, \tag{2.10}$$

which, in fact, results in all of the even-order terms canceling leaving

$$f(x+h)-f(x-h) = 2hf'(x)+2h^3\frac{f^{(3)}}{3!}+\cdots. \tag{2.11}$$

Dropping the higher-order terms and solving for $f'(x)$ gives

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}. \tag{2.12}$$

Since the quadratic terms (actually, all of the even-order terms) subtract out, this approximation is accurate to an additional order compared to the other differencing methods. The improved accuracy of the central difference method is one reason for its use opposed to the simpler forward difference. Another, more prevalent, advantage for central differences is the performance near the optimum. In the area near the minimum, the function naturally takes the form of a quadratic, since the quadratic terms of the Taylor series expansion subtract out, the error of central differences is identically zero near the optimum.

The choice of h here is according to Equation 2.6, just as for forward differences, however a different ϵ is generally used to take into account the additional accuracy of the method and serve as an approximate solution to the trade-off between round-off and truncation error. Here, for central differences,

$\epsilon = 1.0 \times 10^{-4}$ is used unless empirically determined otherwise.

In addition to its added accuracy and favorable near-minimum performance, central differences are still quite simple to implement. For complex problems, though, the runtime can be significantly increased (by approximately a factor of two) as a result of the additional perturbation to the function value that must be determined compared to forward differences. It is then up to the user to decide whether the advantage of additional accuracy outweighs the disadvantage of the time increase in calculating the derivatives. Even then, with precise tuning of the perturbation length and the additional order of accuracy, central differences can still break down in difficult and/or sensitive applications.

2.3 Complex-Step Derivatives

While difference methods are, in general, relatively fast and easy in implement, their downfall, especially in complicated applications, is their accuracy. Several methods have been derived in order to provide more accurate estimates. Automatic differentiation (AD) is a method that takes advantage of the chain-rule property to determine the derivative of a function[10, 11]. Advantages of AD include its ability to calculate derivatives essentially exactly and its extensibility to higher-order derivatives. Disadvantages include a relative difficulty to implement and significantly higher runtimes[6].

Several methods of "analytical" derivative calculation exist which in-

volve relating the derivative to known values and relationships in the problem. A particular example is derived in Section 4.2.1.2. These methods boast derivatives accurate to the numerical error of the coded problem and run faster than any other method, but often require a large manual effort to derive the necessary relationships.

On the other hand, complex-step differentiation is a method that offers the potential for virtually analytic accuracy, with speed and implementation ease on the order of difference methods. CSD can be related to differencing in its derivation, and can be used as an easy implementation of a specific type of AD[12].

Before the general derivation of the CSD approximation is presented here, the relationship can be obtained as a special case of the forward difference approximation given in Equation 2.5. Assuming that $f = u + iv$ is an analytic function of $z = x + iy$ where $i = \sqrt{-1}$, such that $i^2 = -1$, and where

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} &= -\frac{\partial v}{\partial x}, \end{aligned} \tag{2.13}$$

it can be written, using the definition of a derivative given by Equation 2.1 and the first relationship in Equation 2.13, that

$$\frac{\partial u}{\partial x} = \lim_{h \to 0} \frac{v(x + i(y + h)) - v(x + iy)}{h}. \tag{2.14}$$

If the function f is originally real-valued, that is, for $y = 0$,

$$\begin{aligned} z &= x + iy \\ z &= x, \end{aligned} \tag{2.15}$$

so,

$$\begin{aligned} v(x) &= 0 \\ u(x) &= f(x). \end{aligned} \tag{2.16}$$

Thus, Equation 2.14 can be rewritten as

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{\Im\left[f(x + ih)\right]}{h}, \tag{2.17}$$

which, for a small, discrete h gives

$$\frac{\partial f}{\partial x} \approx \frac{\Im\left[f(x + ih)\right]}{h}. \tag{2.18}$$

This gives the estimate of the first derivative of $f(x)$ as the special case of the forward difference approximation of a real-valued, analytic function. This case denotes the limits of the usefulness of a complex-step derivative approximation. The approximation is only valid for real-valued, analytic functions. Now, this is not a significant restriction of the applicability of this method. Most commonly used functions are analytic, though two notable examples of

non-analytic functions are the absolute value function (not analytic at zero) and piecewise-defined functions (not typically analytic where pieces meet).

A more elegant derivation of the complex-step approximation can be obtained similarly to differences, but expanding the Taylor series about a perturbation (h) which lies in the complex plane, that is,

$$f(x+ih) = f(x) + ihf'(x) + (ih)^2\frac{f''(x)}{2} + (ih)^3\frac{f^{(3)}(x)}{3!} + \cdots . \tag{2.19}$$

which, using the relationship $i^2 = -1$, gives

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2} - ih^3\frac{f^{(3)}(x)}{3!} + \cdots . \tag{2.20}$$

Then, separating Equation 2.20 into its real and imaginary parts gives

$$\begin{aligned} \Re[f(x+ih)] &= f(x) - h^2\frac{f''(x)}{2} + h^4\frac{f^{(4)}(x)}{4!} + \cdots \\ \Im[f(x+ih)] &= \Im\left[ihf'(x) - ih^3\frac{f^{(3)}(x)}{3!} - \cdots\right]. \end{aligned} \tag{2.21}$$

Using the imaginary component of Equation 2.21, solving for $f'(x)$ and dropping higher order terms gives

$$f'(x) \approx \frac{\Im[f(x+ih]}{h}, \tag{2.22}$$

which is identical with the expression shown in Equation 2.18. This derivation gives the added effect of showing a key benefit of CSD. Note that there is

no subtraction in this expression, as opposed to the formulas for difference methods. This allows the choice of arbitrarily small values for the perturbation length (h). If the first higher order term dropped is examined, it can be seen that the truncation error can be eliminated if [3]

$$h^2 \left| \frac{f^{(3)}(x)}{3!} \right| < \kappa \left| f'(x) \right| \tag{2.23}$$

where κ is the machine precision. This gives the result that both round-off and truncation error can be eliminated to the accuracy of the algorithm by an appropriately small choice of h.

In addition to accuracy at the level of AD, the single perturbation implies that runtime on the order of forward differences is to be expected, although complications resulting from the use of complex arithmetic are possible. This, however, may be made up for in iterative solutions by the improved accuracy - fewer iterations may be required for convergence. While each iteration may take slightly longer for CSD, the fewer overall iterations may give a reduced overall runtime. It should also be noted here that for environments that do not inherently handle complex arithmetic it may be necessary to include or create the appropriate packages to allow complex values in the functions and routines used to solve a particular problem.

In this work, only the MATLAB environment is used, primarily because its built-in ability to easily handle complex values. There are particular routines and functions in MATLAB that do not accurately compute for com-

plex values, and, for some, no error is given. It is important to make sure that functions used are applicable, and the correct function, for complex-valued inputs. For example, the commonly used 'prime' (') function for the transpose of a matrix or vector actually returns the complex-conjugate transpose, which would result in an incorrect derivative calculation. The correct function for a matrix or vector transpose for complex-valued input is *transpose*. Many examples exist, too many to comprehensively denote here. Shampine (2007)[5] gives several examples, but the user is ultimately responsible for knowing the applicability of complex-valued inputs for their routines and functions.

The aim of the rest of this work is to detail the applicability of CSD for several different methods often used in the solution of optimization problems (Chapter 3) and to explicity show the advantages (and/or disadvantages) of CSD in terms of accuracy, speed, implementation, and robustness as compared to traditional difference methods (Chapter 4)

Chapter 3

Use of CSD in the Lunar Launch Problem

Again, the ability of complex-step derivatives to be used to estimate derivatives has been known for some time. Detailed results from the application of this method to real problems have not been widely published. It is the goal of this chapter to establish the usability of complex-step derivatives for a wide variety of methods for the solution of optimization problems. This is done by comparing the solution of a particular problem, the minimum-time lunar ascent, obtained using traditional differencing methods and complex-step derivatives. Several different tools that may be used in the solution of this problem are tested by this method as well. In particular, the problem is solved using fixed and variable-step integration methods, linear, quadratic, and cubic splines to estimate the control history, using direct optimization methods.

3.1 The Baseline Lunar Ascent Problem

A simple formulation of the lunar ascent problem is the launch of a vehicle from the surface of the moon and its ascent to a particular, given

orbit insertion. The orbit insertion point is defined by an altitude and velocity vector leaving the final downrange position free. A constant thrust acceleration engine is modeled giving the result that the minimum-fuel solution is equivalent to the minimum-time solution.

Some assumptions are made in the solution of this problem. It has been determined by Hull (2010)[13] that the optimal solution for the three-dimensional lunar ascent problem is the case where solution lies entirely in the great-circle plane between the given initial and final conditions. Therefore, the problem modeled here will be strictly the two-dimensional case. Hull also determins that the guidance law derived using an assumption of a flat-moon (a perpendicular, constant gravitational field) is sufficient for the actual spherical-moon problem. Here, a flat-moon gravitational field model is used.

Since, as stated, the minimum-fuel solution is equivalent to the minimum-time solution, the optimization problem will be set up as

$$J = t_f. \tag{3.1}$$

The equations of motion are given by

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u \\ v \\ \alpha \cos\theta \\ \alpha \sin\theta - g \end{bmatrix} \tag{3.2}$$

where $g = 5.32\,\text{ft/s}^2$ and $\alpha = 20.8\,\text{ft/s}^2$. The initial conditions are,

$$\mathbf{X}_0 = \begin{bmatrix} x_0 \\ y_0 \\ u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.3}$$

with final conditions,

$$\mathbf{X}_f = \begin{bmatrix} x_f \\ y_f \\ u_f \\ v_f \end{bmatrix} = \begin{bmatrix} \text{free} \\ 50000\,\text{ft} \\ 5444\,\text{ft/s} \\ 0\,\text{ft/s} \end{bmatrix}. \tag{3.4}$$

Since this problem is being solved with a direct method, the constraints are given by

$$\mathbf{c} = \begin{bmatrix} y(t_f)/y_f - 1 \\ u(t_f)/u_f - 1 \\ v(t_f)/u_f \end{bmatrix} = \mathbf{0}, \tag{3.5}$$

and an optimization parameter vector represented by

$$\mathbf{X}_P = \begin{bmatrix} \theta_i \\ t_f \end{bmatrix} \tag{3.6}$$

where θ_i is a column vector containing the control at each node. The initial

guess for this vector is

$$\mathbf{X}_{P_0} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_0 = \begin{bmatrix} 40^\circ \\ 30^\circ \\ 20^\circ \\ 10^\circ \\ 0^\circ \\ 300\text{s} \end{bmatrix}. \tag{3.7}$$

3.2 Linear Interpolation

The problem is solved assuming a linear interpolated control history with sampling occuring at five nodes given by the scaled values

$$\tau_i = \begin{bmatrix} 0 & 1/4 & 1/2 & 3/4 & 1 \end{bmatrix}, \tag{3.8}$$

where $\tau \triangleq t/t_f$ and with the control at each node denoted by θ_i, representing the pitch angle measured from the horizontal to the thrust vector.

The baseline solution is acheived using a fixed-step, fourth-order Runge-Kutta integrator with twenty steps to integrate the equations of motion in a nonlinear programming (NLP), parameter optimization problem (POP) solved using MATLAB's *fmincon* and its 'sqp' algorithm. Using user-input forward differences to calculate the gradients, where $h_i = \epsilon\,(1 + |X_i|)$ and $\epsilon = 1 \times 10^{-8}$,

the optimal parameter vector is found to be

$$\mathbf{X}_{P_{OPT}} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.040^\circ \\ 20.794^\circ \\ 15.155^\circ \\ 9.197^\circ \\ 3.036^\circ \\ 272.706\text{s} \end{bmatrix}. \tag{3.9}$$

Using the same solution method with the exception of the gradients, the optimal solution is again determined. Here, the gradients are user-input, calculated via complex-step differentiation with a perturbation of $1.0i \times 10^{-14}$. This perturbation length is used because it is lower than the tolerance used in the optimization and the integration, yet is able to be displayed onscreen in MATLAB's format long. Based on the theory reported in Chapter 2, it is expected that the same solution should be reached using complex-step differentiation, with the possibility of small differences arising from more accurate derivative calculation. A more expensive solution should not occur.

The optimal parameter vector for the linear interpolation solution with

complex-step derivatives is given by

$$\mathbf{X}_{P_{OPT}} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.040^\circ \\ 20.794^\circ \\ 15.155^\circ \\ 9.197^\circ \\ 3.036^\circ \\ 272.706\text{s} \end{bmatrix}. \tag{3.10}$$

It can be seen that the exact solution obtained using forward differences is achieved using complex-step derivatives. In addition, convergence was achieved in ten iterations for the complex-step derivative case as opposed to eleven iterations needed for the forward differences. The theory appears to hold for numerical derivatives used in simple optimization algorithms. Several other tools used to solve optimization problems are tested in subsequent sections.

3.3 Higher-Order Spline Interpolation

In the previous solution, the control history is represented by a simple linear interpolation model with five nodes. A continuous control history is likely to be more optimal. In addition, a continuous control differentiable at the node points allows the integration steps to be more widely varied. With a control history, for example a linear interpolation, where the control is not continuous or differentiable across the nodes, the integration steps must coincide

with the nodes to ensure a smooth integration. A continuously differentiable control allows the integration steps to be located at any point along the time history.

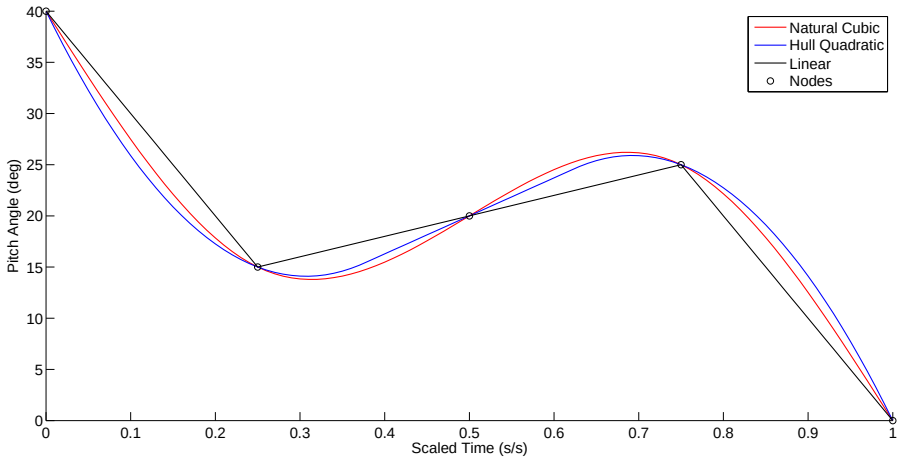


Figure 3.1: Interpolated Control History with Evenly Spaced Nodes and Uneven Control Spacing

Two interpolation schemes are examined to test the use of complex-step derivatives in higher-order spline interpolation. Figure 3.1 shows a contol history defined with linear interpolation, a Hull spline, and a natural cubic spline. The discontinuity in the first derivative of the linearly interpolated control is evident.

3.3.1 Hull Quadratic Spline

The Hull spline is a 2nd order (quadratic) interpolation method first developed by Hull (described via personal communication from October 2010-

April 2011) and loosely based on the Subbotin spline as presented in [14] Section 9.1. To describe the formulation of the spline, nodes (τ_i) are the points at which the values of the function is known, and knots (t_i) are the points where the function of the spline is allowed to change. In this interpolation method, the intial and final times are both nodes and knots, and knots are also placed at the midpoint between interior nodes, that is,

$$\begin{cases} t_0 = \tau_0 \\ t_i = \frac{1}{2}\left(\tau_{i+1} + \tau_i\right), \quad 1 \leq i \leq n-1 \cdot \\ t_n = \tau_{n+1} \end{cases} \tag{3.11}$$

The benefit of the Hull spline is that it is a simple, continuous interpolation method that leaves no free parameters. The downside of the Subbotin spline is that the knots must be defined in order to determine the node placement. In many real problems, the placement of the knots is not significant, but, since the function is only known at certain places, the nodes are constrained. As a result of this, the Hull spline was derived to provide a quadratic interpolation with knots determined from defined nodes. Between knots, the shape of the spline function is given by

$$S_i = y_{i+1} + \frac{1}{2}\left(z_{i+1} + z_i\right)\left(\tau - \tau_{i+1}\right) + \frac{1}{2h_i}\left(z_{i+1} - z_i\right)\left(\tau - \tau_{i+1}\right)^2 \tag{3.12}$$

with y_i representing the value at each node (in this case, θ), and

$$h_i = \frac{1}{2}\left(\tau_{i+1} - \tau_i\right). \tag{3.13}$$

The value z_i, representing the continuous slope at the knot, is found by solving the linear system of equations given by

$$\begin{bmatrix} 8 & 0 & 0 & 0 & 0 \\ 1 & 6 & 1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 1 & 6 & 1 \\ 0 & 0 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \\ z_n \end{bmatrix} = 8 \begin{bmatrix} \left(y_1 - y_0\right)/h_0 \\ \left(y_2 - y_1\right)/h_1 \\ \vdots \\ \left(y_n - y_{n-1}\right)/h_{n-1} \\ \left(y_{n+1} - y_n\right)/h_n \end{bmatrix}. \tag{3.14}$$

In the solution of the Lunar Launch problem using a Hull Spline for the control history, the same initial guess and model is used as with the baseline. This gives a slightly different solution than in the baseline, but this is a result of the difference in the interpolation as the controls iterate toward the optimum. The optimal parameter vector is identical for both solution methods and is

shown by

$$\mathbf{X}_{P_{OPT}}|_{HQS} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.007^\circ \\ 20.762^\circ \\ 15.129^\circ \\ 9.180^\circ \\ 3.021^\circ \\ 272.706\text{s} \end{bmatrix}, \tag{3.15}$$

In both cases, the solution converged to the accepted solution in nine iterations. Just as in the linearly interpolated case, identical solutions are achieved between the two differentiation methods.

3.3.2 Natural Cubic Spline

Cubic splines are commonly used for interpolation and function approximation because they are continuous functions and their first and second derivatives are continuous as well. This is of particular benefit for applications where second-derivatives are necessary. In addition, the continuity of the second derivative adds the aesthetic benefit of the spline appearing smooth. Finally, cubic splines are most often used because they not only allow for continuity in the first and second derivatives but also are more resistant to oscillations that may be found using higher-order polynomials. The cubic spline used here is derived in [14], Section 9.2.

As opposed to the Hull spline, a natural cubic spline requires the defi-

nition of the nodes which then coincide with the knots. Here, the same node distribution is used as in the baseline model (given by Equation 3.8). Again, the same intial guess and model is used as in all previous trials. The optimal parameter vector for both methods is given by

$$\mathbf{X}_{P_{OPT}}|_{NCS} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.025^\circ \\ 20.765^\circ \\ 15.126^\circ \\ 9.183^\circ \\ 3.028^\circ \\ 272.706\text{s} \end{bmatrix}, \tag{3.16}$$

Once again, the two methods of gradient determination give an identical result to each other, and once again the solution is near, but not identical to those found using the other interpolation methods. Here, convergence to the accepted solution is achieved in ten iterations for both methods.

3.4 Integration Methods

When using a linearly interpolated control history, it is important to coordinate the nodes with integration steps. If this is not done, the integrator will be trying to integrate over a function where the first derivative is discontinuous when the node lies in the middle of an integration step. Depending on the function, this may cause large error in the integration leading to inaccura-

cies or even failure to converge or solve. This is, perhaps, the most important consideration for using spline interpolation: to establish a continuous function over which to integrate. Having a continuous function allows the integration steps to be placed arbitrarily without having to worry about discontinuities. The allowance of arbitrary integration step placement leads to the use of variable step-size integration, since, in most cases, the length of the step is left for the integrator to determine. Now, no restrictions on step length or placement exist. In sensitive problems, the use of a variable step-size integrator can greatly reduce the integration error with appropriate step placement; short steps in sensitive areas to maintain accuracy, and longer steps in less sensitive areas to aid speed.

Here, both spline functions, Hull and natural cubic, are tested with a variable step-size integrator: MATLAB's *ode45* used with the relative and absolute tolerances both set to 1×10^{-10} to match the tolerances set in the optimizer. For both spline functions, the solution process is identical to that described in their respective sections (Sections 3.3.1 and 3.3.2).

The optimal parameter vector for the solution of the lunar ascent prob-

lem using variable-step integration and a Hull spline is given by

$$\mathbf{X}_{P_{OPT}}|_{FD} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.007^\circ \\ 20.761^\circ \\ 15.129^\circ \\ 9.180^\circ \\ 3.020^\circ \\ 272.706\text{s} \end{bmatrix} \tag{3.17}$$

for forward differences, and by

$$\mathbf{X}_{P_{OPT}}|_{CS} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix}_{OPT} = \begin{bmatrix} 26.007^\circ \\ 20.762^\circ \\ 15.129^\circ \\ 9.180^\circ \\ 3.021^\circ \\ 272.706\text{s} \end{bmatrix} \tag{3.18}$$

for complex-step derivatives. Here, the results for forward differences and complex-step differentiation are not exactly the same. A nearly identical result is shown here for forward differences as was found for the fixed-step integration with a Hull spline (given by Equation 3.15), and the complex-step derivatives gives the same result as the fixed-step integration. The forward difference case also converges in twenty-two iterations and 81.679 seconds while the complex-

step differentiation case converges in ten iterations and 12.470 seconds. The difference in the convergence statistics as well as the slight difference in the result is most likely caused by more accurate calculation of the gradient.

The optimal parameter vector for the solution of this problem using variable-step integration and a natural cubic spline is given by

$$\mathbf{X}_{P_{OPT}}|_{FD} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ t_f \end{bmatrix} = \begin{bmatrix} 26.025^\circ \\ 20.765^\circ \\ 15.126^\circ \\ 9.183^\circ \\ 3.028^\circ \\ 272.706\text{s} \end{bmatrix}, \tag{3.19}$$

Again, the solution for each gradient method matches up identically here, as well as with with the fixed step-size integration results (Equation 3.16). It should also be noted that in all of the previous solutions the optimization process converged in virtually the same amount of time, and identically the same number of iterations with the exception of the variable step-size, natural cubic spline result. Here, the complex-step solution converged in ten iterations and 30.387s, while the forward difference solution converged in fifteen iterations and 137.903s. This provides another example of the advantages of more accurate numerical derivatives, advantages that are examined more closely in Section 4.3.

Chapter 4

Analysis of CSD in the Orbit Transfer Problem

In addition to determining the applicability of complex-step differentiation to different features and methods used to solve constrained optimization problems, as seen in Chapter 3, it is desired to compare this differentiation method to others in a numerical sense. As the solution of the Lunar Launch Problem is easily varied to include several different features and methods commonly seen in optimization problem, a new problem, the Orbit Transfer problem, is used as the baseline for the numerical analysis. In Section 4.1 a thorough description of the general and specific aspects of this problem is given. Section 4.2 details the solution method to the problem as well as the means of handling different versions and the application of CSD to the solution. Finally, Section 4.3 explains the methods of comparison and analysis and gives the numerical results and their implications.

4.1 The Optimal Orbit Transfer Problem

Simply put, the optimal Orbit Transfer Problem (OTP) usually involves finding the minimum time or minimum fuel solution to a spacecraft transfer

between two distinct orbits of a central body. In this case the minimum fuel solution is the interest, and the central body is the Earth which is assumed to be an inertial reference frame. This problem is of particular interest since the primary constraint in space missions is monetary cost and the cost of a mission can be most easily reduced by minimizing the amount of fuel necessary to complete the mission. This is a because the primary cost of a space mission is launching mass out of the Earth's gravitational well. Reducing the amount of fuel needed not only decreases the mass of the spacecraft but reduces its volume as well, thereby reducing the necessary structural mass.

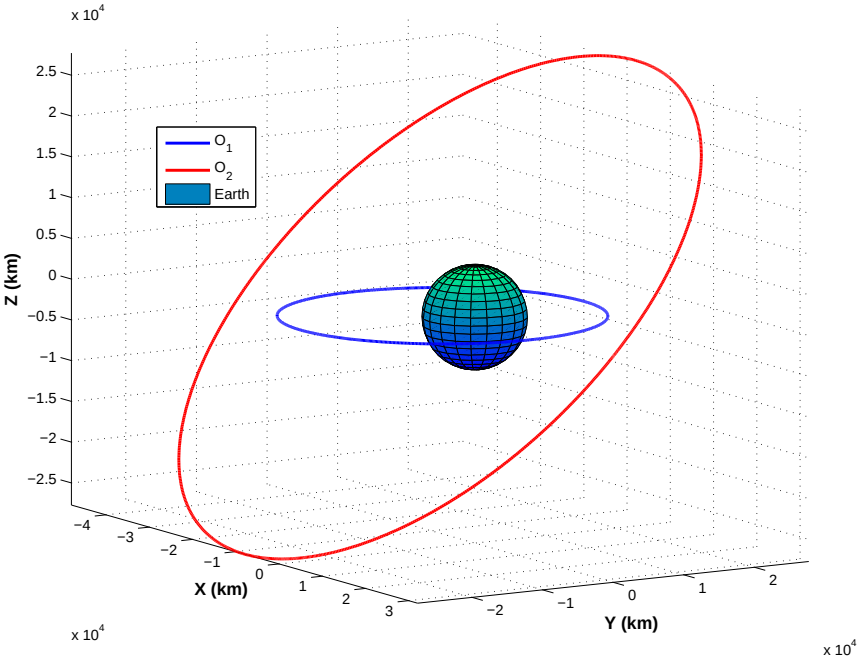
In its most simple sense, the solution to the OTP is generally regarded to be a Hohmann transfer-an 180°, two-impulse transfer[15]. This solution, however, requires several assumptions including coplanar, aligned orbits and impulsive burns at the beginning and end-points. In some cases, a three-impulse maneuver (known as a Bi-Elliptic transfer) can actually require less fuel than a Hohmann transfer, but this often requires a very large flight time[16]. Here, more realistic assumptions are made. The orbits are not constrained to be coaxial or coplanar, and, in addition to impulsive burns, solutions with finite burn arcs are assessed.

As mentioned, the OTP is not constrained to coplanar, coaxial orbits. Specifically, a transfer between two elliptical orbits of different sizes where the orbits are inclined and rotated with respect to each other is considered. The two orbits are defined by their classical orbital elements as seen in Table 4.1.

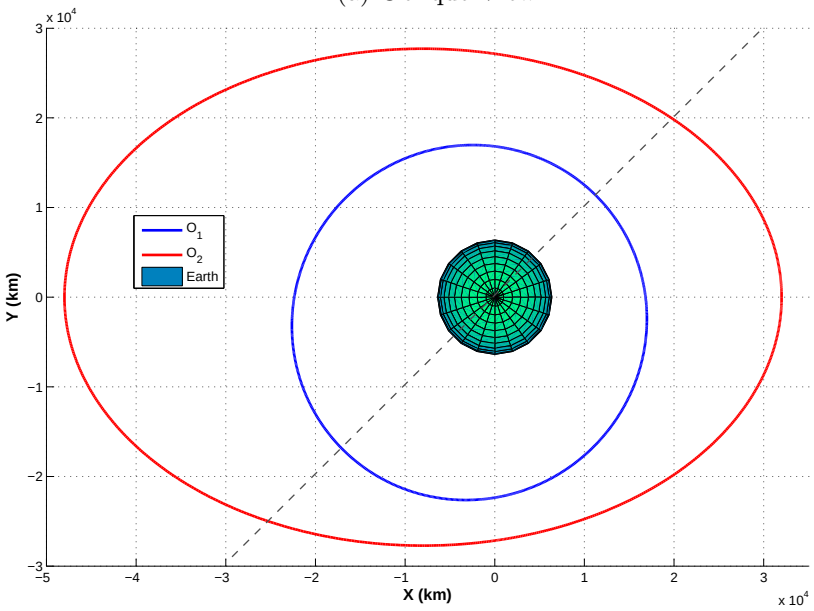
Table 4.1: Initial and Final Orbits

Element	Orbit 1	Orbit 2
a (km)	20000	40000
e	0.2	0.2
i	0	45°
Ω	0	0
ω	45°	0

Figure 4.1 shows the two orbits around the Earth. Figure 4.1a is an oblique view of the two orbits emphasizing the inclination change while Figure 4.1b is a top view (of the X-Y plane) to indictate the change in argument of periapse. The dashed line is the apseline of the smaller orbit (Orbit 1) while the X-axis represents the apseline of the larger orbit (Orbit 2).



(a) Oblique View



(b) Top View of XY Plane

Figure 4.1: Spatial Views of Orbits 1 and 2 About the Earth

For this problem, a constant set of spacecraft and gravitational field parameters are assumed. An Earth-centered reference frame with a simple 2-body, inverse-square gravitational field is modeled and the standard value for the gravitational parameter ($\mu_E = 398600.8^{\mathrm{km}^3/\mathrm{s}^2}$) for Earth is used. The parameters for the spacecraft engine are found in Table 4.2.

Table 4.2: Spacecraft Parameters

Parameter	Value
m_0	1000 kg
T_{min}	0.0 kN
T_{max}	0.1 kN
I_{sp}	1000 s
g_0	0.009806 $^{\mathrm{km}/\mathrm{s}^2}$

4.2 Solution of the OTP

The first solution of this problem is the impulsive-maneuver solution described in Section 4.2.1. An instantaneous burn is applied at a point to be determined on the first orbit, the spacecraft coasts on a ballistic trajectory until intercept of the second orbit, and a second instantaneous burn is applied to match the velocity at that point.

The second solution that is considered is a segmented, finite-burn solution as described in Section 4.2.2. The thrust direction of the spacecraft is assumed to constant for each segment but can change between successive segments. The length and direction of the burn arc is dependent on the thrusting ability of the engine. The number of overall segments are prescribed for

a given solution. Three and five-segmented solutions are analyzed. In general, the optimal solution begins and ends with a burn arc with a coast arc in between.

4.2.1 Impulsive Solution

To numerically solve the impulsive-maneuver optimal Orbit Transfer Problem, a nonlinear programming (NLP), parameter optimization problem (POP) is formed. This requires definition of an objective function, optimization parameters, constraints, and nonlinear equations of motion. As stated, the equations of motion are based on a 2-body, inverse-square gravitational law.

4.2.1.1 Optimization Setup

The state vector is defined in Equation 4.1 and Equation 4.2 gives the equations of motion.

$$\mathbf{X} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ m \end{bmatrix} \tag{4.1}$$

$$\dot{\mathbf{X}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{r^3}\mathbf{r} \\ -\frac{T}{c} \end{bmatrix} \tag{4.2}$$

Here, $c = g_0 I_{sp}$.

The integration of the equations of motion is performed using MATLAB's *ode45*, a variable step-size integrator where the absolute and relative tolerances at each step are set to 1×10^{-12}. In the case, described below, where the State Transition Matrix (STM) is used to calculate the first-order Jacobian, the same settings are used for its integration as well.

For this problem, the spacecraft's initial position is given to be on Orbit 1 (location on the orbit is free), and its final position is constrained to be on Orbit 2 (again, the location on the orbit is free). This constraint is achieved without needing to determine the orbital elements of the spacecraft at every iteration step. The five linearly independent orbital elements are replaced by the 5×1, linearly independent vector made up of the angular momentum and x and y components of the eccentricity vector. The constraint vector is thus given by

$$\mathbf{c}(\mathbf{X}_P) = \begin{bmatrix} \mathbf{h}(t_f) - \mathbf{h}^* \\ e_x(t_f) - e_x^* \\ e_y(t_f) - e_y^* \end{bmatrix} = \mathbf{0} \tag{4.3}$$

where $\mathbf{h} = \mathbf{r} \times \mathbf{v}$, and $\mathbf{e} = {}^{\mathbf{v}\times\mathbf{h}}/_{\mu} - {}^{\mathbf{r}}/_{\|\mathbf{r}\|}$.

Here, the starred values represent the prescribed angular momentum and eccentricity of Orbit 2 which are subtracted from the actual value of the spacecraft at the endpoint. The constraint vector is forced to zero by the

optimizer with a tolerance set to 1×10^{-12}. In addition, t_2, the time of the second impulsive burn, is constrained to be greater than t_1, the time of the first impulsive burn, ensuring a positive time of flight.

An optimization parameter vector, $\mathbf{X}_P$, is chosen to fully define the problem. $\mathbf{X}_P$ is shown by

$$\mathbf{X}_P = \begin{bmatrix} t_1 \\ \Delta\mathbf{v}_1 \\ t_2 \\ \Delta\mathbf{v}_2 \end{bmatrix}. \tag{4.4}$$

This parameter vector is chosen because it allows the locations on Orbits 1 and 2 to remain optimization variables as well as letting the impulsive velocity change at each burn ($\Delta\mathbf{v}$) to vary as well. The times t_1 and t_2 each are measured from a specified $t_0 = 0$, defined as the periapse passage on Orbit 1.

With the parameters, constraints, and state defined, the optimization process is performed. Since the minimum fuel solution is desired, an objective function is so defined. In the impulsive solution, fuel is represented as $\Delta v = \|\Delta\mathbf{v}\|$, the change in velocity. The objective function is then defined to be the sum of the magnitudes of the change in velocity vectors at each burn point, that is,

$$J = \min \langle \|\Delta\mathbf{v}_1\| + \|\Delta\mathbf{v}_2\| \rangle . \tag{4.5}$$

MATLAB's *fmincon* optimization function is used with its 'sqp' optimization algorithm to solve the NLP POP. This program is chosen for its ability to solve constrained optimization problems combined with its ease in handling complex numbers. Another important reason for the use of *fmincon* is its ability to either compute its own or to allow user-input partials. First derivatives of both the objective function and the constraint function with respect to the parameter vector are required if the user is going to supply the Jacobians. This is done thrice, as the problem is solved separately using user-input partials via three methods: Central Differences, Linear Perturbation Analysis (or State Transition Matrix derivatives), and Complex-Step Differentiation.

4.2.1.2 Linear Perturbation Analysis

While central differences and complex-step differentiation have been detailed in Chapter 2, the third method has not. This method for computing the first-order partials of the objective function and the constraint equations is Linear Perturbation Analysis [17]. This method determines the derivatives analytically using the State Transition Matrix. Since the derivatives are analytical, this method is used as the comparison basis. Even though the derivatives are analytical, this method is not strictly exact; there is usually some error in the STM as a result of the numerical integration. This is not a major factor in the comparison, because both of the other methods also go through numerical integration with the same order of accuracy.

For this problem, the STM is numerically integrated simultaneously with the equations of motion, subject to the initial condition that $\mathbf{\Phi}(t_0, t_0) = \mathbf{I}$. The rate of change of $\mathbf{\Phi}$ is denoted by

$$\dot{\mathbf{\Phi}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \mathbf{\Phi} \tag{4.6}$$

where $\mathbf{G}$ is the Gravity Gradient Matrix, defined for the three degree-of-freedom, 2-body problem by

$$\mathbf{G} = \begin{bmatrix} \frac{2\mu(x^2 - 0.5(y^2+z^2))}{r^5} & \frac{3\mu xy}{r^5} & \frac{3\mu xz}{r^5} \\ \frac{3\mu xy}{r^5} & \frac{2\mu(y^2 - 0.5(x^2+z^2)}{r^5} & \frac{3\mu yz}{r^5} \\ \frac{3\mu xz}{r^5} & \frac{3\mu yz}{r^5} & \frac{2\mu(z^2 - 0.5(x^2+y^2)}{r^5} \end{bmatrix}. \tag{4.7}$$

Once the STM is obtained, it is necessary to derive formulas for the derivatives of the objective function and constraints with respect to the parameter vector in terms of the STM. Since the derivatives of the objective function and the inequality constraints are simpler and are available analytically, they are input. The partials of the equality constraints, however, are derived as

$$\frac{\partial \mathbf{c}}{\partial \mathbf{X}_P} = \frac{\partial \mathbf{c}}{\partial \mathbf{r}(t_2)} \frac{\partial \mathbf{r}(t_2)}{\partial \mathbf{X}_P} + \frac{\partial \mathbf{c}}{\partial \mathbf{v}(t_2)} \frac{\partial \mathbf{v}(t_2)}{\partial \mathbf{X}_P}. \tag{4.8}$$

Then, assuming that $\mathbf{e} = \begin{bmatrix} e_x \\ e_y \end{bmatrix}$,

$$\begin{aligned} \frac{\partial \mathbf{c}}{\partial \mathbf{r}(t_2)} &= \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{r}(t_2)} \\ \frac{\partial \mathbf{e}}{\partial \mathbf{r}(t_2)} \end{bmatrix} \\ \frac{\partial \mathbf{c}}{\partial \mathbf{v}(t_2)} &= \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{v}(t_2)} \\ \frac{\partial \mathbf{e}}{\partial \mathbf{v}(t_2)} \end{bmatrix}. \end{aligned} \tag{4.9}$$

These terms can be shown to be

$$\begin{aligned} \frac{\partial \mathbf{h}}{\partial \mathbf{r}(t_2)} &= \mathbf{I} \times \mathbf{v}(t_2) \\ \frac{\partial \mathbf{e}}{\partial \mathbf{r}(t_2)} &= \frac{1}{\mu} \left(\mathbf{v}(t_2) \times (\mathbf{I} \times \mathbf{v}(t_2)) \right) - \left(\frac{1}{r} \mathbf{I} - \frac{\mathbf{r}(t_2)\mathbf{r}^{\mathrm{T}}(t_2)}{r^3} \right) \\ \frac{\partial \mathbf{h}}{\partial \mathbf{v}(t_2)} &= \mathbf{r}(t_2) \times \mathbf{I} \\ \frac{\partial \mathbf{e}}{\partial \mathbf{v}(t_2)} &= \frac{1}{\mu} \left(\mathbf{I} \times \mathbf{h} + \mathbf{v}(t_2) \times (\mathbf{r}(t_2) \times \mathbf{I}) \right) \end{aligned} \tag{4.10}$$

where $\mathbf{I}$ represents a 3×3 identity matrix.

The $\partial \mathbf{r}(t_2) / \partial \mathbf{x}_P$ and $\partial \mathbf{v}(t_2) / \partial \mathbf{x}_P$ terms can be derived to show that [18]

$$\begin{aligned}
\frac{\partial \mathbf{r}(t_2)}{\partial t_1} &= -\mathbf{\Phi}_{11}\Delta\mathbf{v}_1 \\
\frac{\partial \mathbf{r}(t_2)}{\partial \Delta\mathbf{v}_1} &= \mathbf{\Phi}_{12} \\
\frac{\partial \mathbf{r}(t_2)}{\partial t_2} &= \mathbf{v}(t_2^-) \\
\frac{\partial \mathbf{r}(t_2)}{\partial \Delta\mathbf{v}_2} &= \mathbf{0}
\end{aligned} \tag{4.11}$$

and

$$\begin{aligned}
\frac{\partial \mathbf{v}(t_2)}{\partial t_1} &= -\mathbf{\Phi}_{21}\Delta\mathbf{v}_1 \\
\frac{\partial \mathbf{v}(t_2)}{\partial \Delta\mathbf{v}_1} &= \mathbf{\Phi}_{22} \\
\frac{\partial \mathbf{v}(t_2)}{\partial t_2} &= -\frac{\mu}{r^3}\mathbf{r}(t_2) \\
\frac{\partial \mathbf{v}(t_2)}{\partial \Delta\mathbf{v}_2} &= \mathbf{I}.
\end{aligned} \tag{4.12}$$

where $\mathbf{\Phi} = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix}$ and the superscript '-' represents the instant before the impulse is applied.

Combining the results in Equations 4.10, 4.11, and 4.12 gives the 8×5 matrix Jacobian for the equality constraint vector $\mathbf{c}(\mathbf{X}_P)$ with respect to the paramter vector $\mathbf{X}_P$ denoted in 4.8.

4.2.1.3 Solution

MATLAB's *fmincon* requires an intial guess for the parameter vector to start its iteration process. The closer the initial guess is to the actual solution the faster the optimizer converges. For this problem, the same initial guess is used for each derivative method.

The initial guess of $\Delta\mathbf{v}_1$ and $\Delta\mathbf{v}_2$ is determined by approximating the transfer trajectory as an 180-degree transfer, neglecting both the change in inclination and the change in argument of periapse. The time of flight is found by approximating an 135-degree transfer, taking into account the change in argument of periapse but neglecting the inclination change. The time of the first burn, t_1, is set equal to $t_0 = 0$, therefore t_2 is the estimated time of flight. After finding a local optimal solution near this location, the initial guess is changed, and a second, local optimal solution was found. The new initial guess is determined by rotating the first guess approximately halfway around the orbit and determining the new $\Delta\mathbf{v}_1$and $\Delta\mathbf{v}_2$. The two successful inital

guesses are given by

$$\mathbf{X}_P = \begin{bmatrix} t_1 \\ \Delta\mathbf{v}_1 \\ t_2 \\ \Delta\mathbf{v}_2 \end{bmatrix}, \quad \mathbf{X}_{0_1} = \begin{bmatrix} 0 \\ -0.654 \\ 0.654 \\ 0 \\ 21363 \\ 0 \\ -1.6 \\ 0 \end{bmatrix}, \quad \mathbf{X}_{0_2} = \begin{bmatrix} 10000 \\ 0 \\ -1.654 \\ -0.2 \\ 31363 \\ 1.0 \\ 0 \\ 1.0 \end{bmatrix}. \tag{4.13}$$

As stated , two locally optimal solutions are found roughly 180° apart in the inertial reference frame. The optimal parameter vectors are

$$\mathbf{X}_P = \begin{bmatrix} t_1 \\ \Delta\mathbf{v}_1 \\ t_2 \\ \Delta\mathbf{v}_2 \end{bmatrix}, \quad \mathbf{X}_{OPT_1} = \begin{bmatrix} -3683.9 \\ 0.2332 \\ 0.6787 \\ 0.3126 \\ 25334 \\ 0.1972 \\ 0.2704 \\ -1.7118 \end{bmatrix}, \quad \mathbf{X}_{OPT_2} = \begin{bmatrix} 10616.0 \\ 0.1657 \\ -0.5528 \\ -0.8616 \\ 33506.8 \\ 0.2954 \\ -0.4367 \\ 2.1173 \end{bmatrix}. \tag{4.14}$$

The total Δv for the first solution is $2.527^{\mathrm{km}}/_{\mathrm{s}}$, and the total Δv for the sec-

ond solution is $3.072^{\mathrm{km}}/_{\mathrm{s}}$. Figure 4.2 shows the transfer trajectory for the lower of the two local optima, Figure 4.3 shows the transfer trajectory for the higher of the two. The first trajectory results in a final mass of the spacecraft of $772.83^{\mathrm{km}}/_{\mathrm{s}}$ while the second trajectory gives a final spacecraft mass of $731.05^{\mathrm{km}}/_{\mathrm{s}}$.

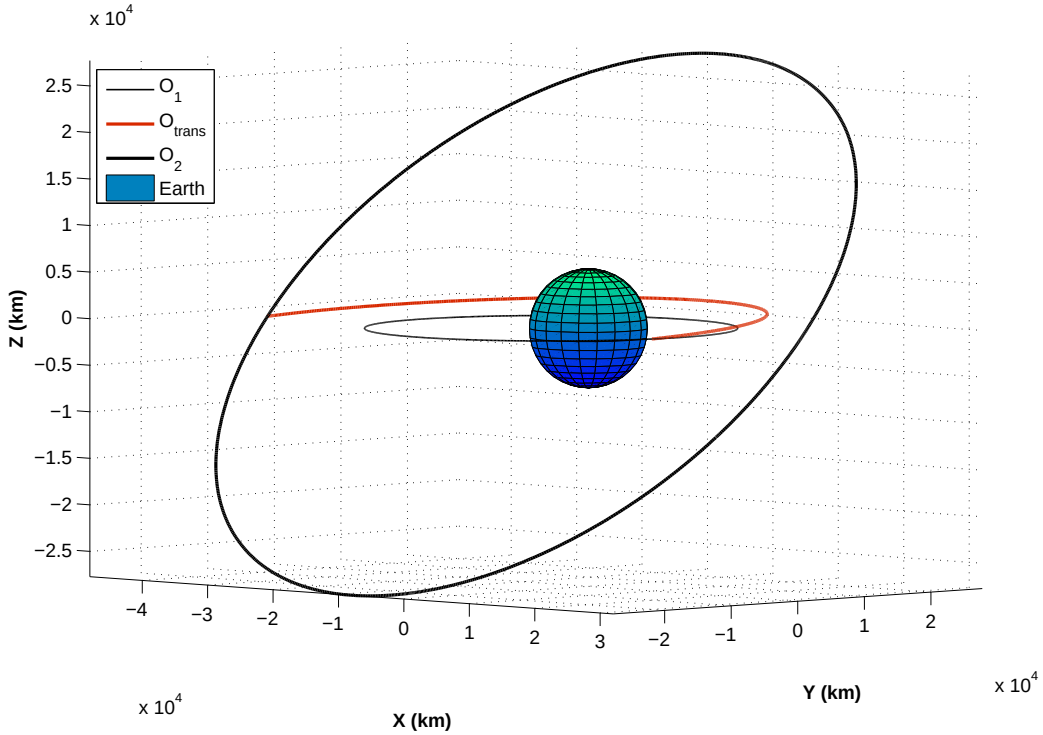


Figure 4.2: Locally Optimal Solution, $\Delta v = 2.527\frac{\mathrm{km}}{\mathrm{s}}$

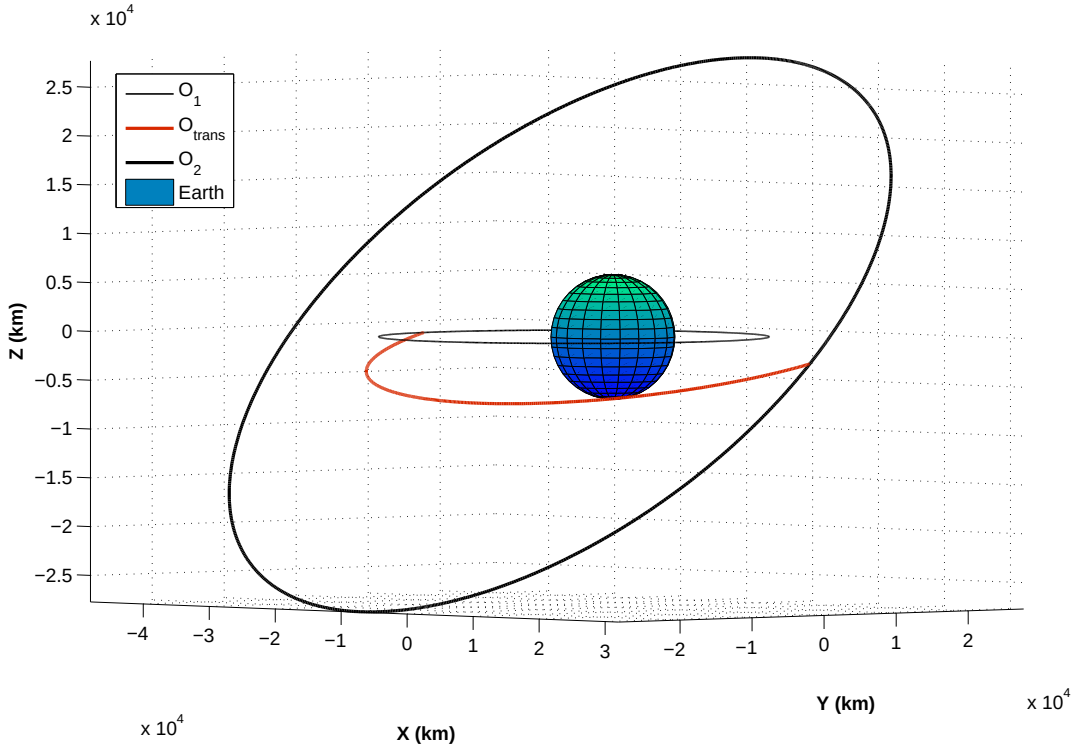


Figure 4.3: Locally Optimal Solution, $\Delta v = 3.072 \frac{\text{km}}{\text{s}}$

4.2.2 Finite-Burn Solution

A more realistic scenario, since current engines cannot achieve an instantaneous burn, is modeling the transfer with a series of separate segments of finite length. Each segment is classified as a "burn arc" or a "coast arc," where burn arcs represent a segment of the mission where the spacecraft engine is on, and, conversely, coast arcs represent a segment of the mission where the spacecraft engine is off. Generally, the solution to this problem is acheived by presupposing a number of segments and allowing the optimizer to determine whether each segment is a burn arc or a coast arc. Also generally speaking, for

relatively high-thrust engines the optimal solution consists of one or multiple burn arcs at the beginning followed by a coast arc with one or multiple burn arcs to complete the transfer.

In Section 4.2.2.1 a three-segment transfer is assumed, which should result in a burn-coast-burn transfer. Then, in Section 4.2.2.2 a five segment solution is assumed, implying a burn-burn-coast-burn-burn result. It should be noted that for each burn arc, the thrust control vector is constrained to be constant.

To numerically solve the Finite-Burn Optimal Orbit Transfer Problem the NLP POP is formed. This requires definition of the objective function, optimization parameters, the constraints, and the nonlinear equations of motion. As stated, the equations of motion are based on a 2-body, inverse-square gravitational law. As a result of the similarity between this problem and the impulsive-maneuver solution, many components could be either copied exactly or with minor changes. Each finite-burn solution is determined using complex-step differentiation and using user-input central differences. The added complexity of this version of the optimal Orbit Transfer Problem allows the differences in the gradients to have an additional effect., Here the optimization converges to the same solution, but a significantly different number of iterations is required.

4.2.2.1 Three-Segment Solution

The setup for the three-segment solution of the Optimal Orbital Transfer is based on the solution for the impulsive problem. Differences occur from the addition of a thrusting term in the equations of motion. The state vector is the same as given in Equation 4.1. The dynamical equations, however, account for the thrust of the spacecraft. For this problem, they are given by

$$\dot{\mathbf{X}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{r^3}\mathbf{r} + \frac{T}{m}\mathbf{u} \\ -\frac{T}{c} \end{bmatrix}. \tag{4.15}$$

The thrust acceleration term consists of the thrust magnitude scaled by the spacecraft mass, and is pointed along a control vector, $\mathbf{u}$, which is constrained in this problem to be constant for each segment. Since this is a three-dimensional problem, $\mathbf{u}$ is a 3×1 vector. To decrease the number of parameters necessary to solve the problem $\mathbf{u}$ is defined by Equation 4.16 in terms of the spherical angles α and β, as

$$\mathbf{u} = \begin{bmatrix} \cos(\alpha)\cos(\beta) \\ \sin(\alpha)\cos(\beta) \\ \sin(\beta) \end{bmatrix}. \tag{4.16}$$

Here, α represents the right ascension and β represents the inclination of a unit vector in a spherical coordinate frame as seen in Figure 4.4.

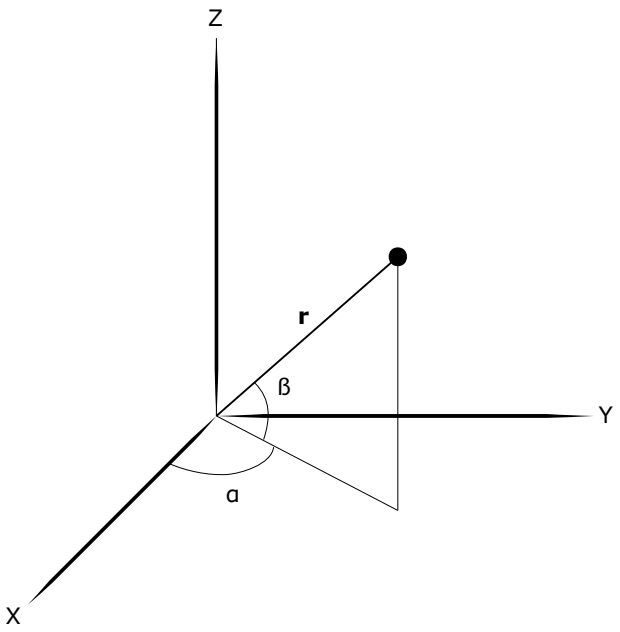


Figure 4.4: Spherical Angles α and β.

From here, the optimization parameter vector is defined to be made up of the times indicating the start of the transfer, the end of the first segment, the end of the second segment, and the end of the third segment (the end of the transfer), as well as the control (T,α,β) for each segment. This results in a 13×1 parameter vector shown in Equation 4.17.

$$\mathbf{X}_P = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ \alpha_1,\beta_1 \\ \alpha_2,\beta_2 \\ \alpha_3,\beta_3 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{4.17}$$

The constraint vector for this solution remains the same as in Equation 4.3. The additional flight times here are still constrained to be positive, and each thrust value is constrained to be between the spacecraft engine T_{min} and T_{max} giving an inequality constraint vector of

$$\mathbf{d}(\mathbf{X}_P) = \begin{bmatrix} t_2 \geq t_1 \\ t_3 \geq t_2 \\ t_4 \geq t_3 \end{bmatrix}. \tag{4.18}$$

Even though the minimum fuel solution is still desired, the objective function for this problem changes to account for the finite burn aspect. Here, the objective function is based on a summation of the used fuel for each flight

segment, that is,

$$J = -\left(m_0 - \sum_{i=1}^{3} \frac{T_i}{c}(t_{i+1} - t_i) \right). \tag{4.19}$$

The initial guess for the optimization of this problem is based on the solution of the impulsive problem. The initial and final burn time from the impulsive solution are taken as the center of the initial and final segment of this solution. The length of each finite burn is determined by converting the known Δv from each burn of the impulsive solution into the time of flight necessary to consume that amount of fuel via the Tsiolkovsky Rocket Equation

$$\Delta v = I_{sp} g_0 \ln \left| \frac{m_0}{m_f} \right|. \tag{4.20}$$

The middle segment is guessed to be a coast arc, and each burn arc is guessed to occur at T_{max}. The angles for the first and last segment are determined from the direction of the impulsive $\Delta \mathbf{v}$. Since the middle segment is guessed to be a coast arc, the angles for this segment do not matter and are guessed to be (0,0). Putting all of this together, the initial guess for the parameter vector is shown by

$$\mathbf{X}_{P_0} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ \alpha_1,\beta_1 \\ \alpha_2,\beta_2 \\ \alpha_3,\beta_3 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix}_0 = \begin{bmatrix} -7421.5 \\ 54.6 \\ 17957.5 \\ 32710.5 \\ -1.24, 0.4108 \\ 0.0, 0.0 \\ 0.9407, -1.3777 \\ 0.1 \\ 0.0 \\ 0.1 \end{bmatrix}. \tag{4.21}$$

Both CSD and user-input central differences are used to optimize this problem using the same function, algorithm, and settings as for the impulsive solution. Both methods converge to the same solution. The optimal parameter vector is given by

$$\mathbf{X}_{P_{OPT}} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ \alpha_{1,}\beta_1 \\ \alpha_{2,}\beta_2 \\ \alpha_{3,}\beta_3 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix}_{OPT} = \begin{bmatrix} -8405.5 \\ 233.2 \\ 17483.1 \\ 32910.6 \\ -1.899, 2.904 \\ 0.0, 0.0 \\ 4.198, -1.741 \\ 0.1 \\ 0.0 \\ 0.1 \end{bmatrix}, \tag{4.22}$$

and the trajectory is shown in Figure 4.5. The optimal final mass calculated for this solution is $m_f = 754.8$kg.

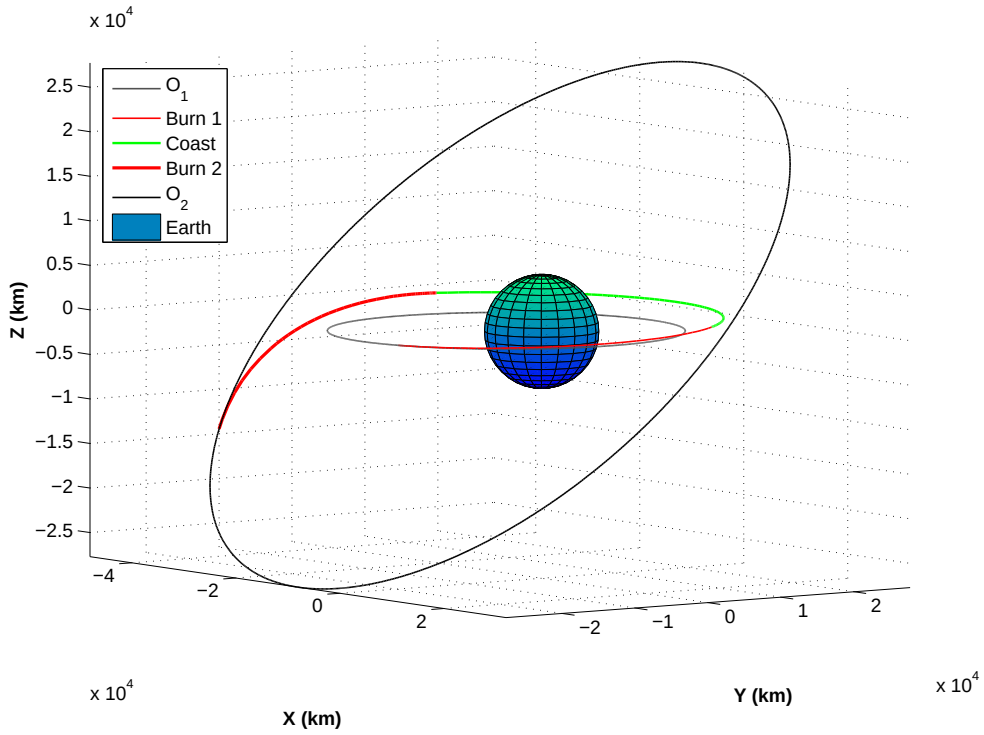


Figure 4.5: Minimum-Fuel Three-Segment Solution

4.2.2.2 Five-Segment Solution

The Optimal Orbit Transfer Problem was also solved for the case of five separate segments. Setup for this solution is similar to the previous three-segment and impulsive burn scenarios. The parameter vector here is expanded to include the angles, times, and thrust magnitudes for the additional segments and is shown in Equation 4.23 along with the initial guess

$$\mathbf{X}_P = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ \alpha_1,\beta_1 \\ \alpha_2,\beta_2 \\ \alpha_3,\beta_3 \\ \alpha_4,\beta_4 \\ \alpha_5,\beta_5 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix}_0 = \begin{bmatrix} -7421.6 \\ -3684.0 \\ 54.6 \\ 17957.5 \\ 25334.0 \\ 32710.5 \\ -1.24, 0.4108 \\ -1.24, 0.4108 \\ 0.0, 0.0 \\ 0.9407, -1.3777 \\ 0.9407, -1.3777 \\ 0.1 \\ 0.1 \\ 0.0 \\ 0.1 \\ 0.1 \end{bmatrix}. \tag{4.23}$$

The initial guess for the five-segment solution is obtained from the successful intial guess used in the three-segment case. The initial and final burn arcs are split in half to give the additional two burn arcs in the burn-burn-coast-burn-burn assumed trajectory. For the initial guess, the direction of the thrust is guessed to be the same for the intial two burn arcs and for the final two burn arcs.

This initial guess converges using both CSD and central differencing to the optimal parameter vector given by

$$\mathbf{X}_{P_{opt}} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ \alpha_1,\beta_1 \\ \alpha_2,\beta_2 \\ \alpha_3,\beta_3 \\ \alpha_4,\beta_4 \\ \alpha_5,\beta_5 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix}_{OPT} = \begin{bmatrix} -7893.7 \\ -3829.3 \\ 46.7 \\ 17608.5 \\ 25336.0 \\ 33705.3 \\ 0.8691, 0.2811 \\ 1.6066, 0.3585 \\ 0.0, 0.0 \\ 1.3442, -1.3631 \\ 0.8451, -1.3768 \\ 0.1 \\ 0.1 \\ 0.0 \\ 0.1 \\ 0.1 \end{bmatrix}, \tag{4.24}$$

and to the trajectory in Figure 4.6. The final mass of the spacecraft for the five-segment trajectory is $m_f = 761.5$kg.

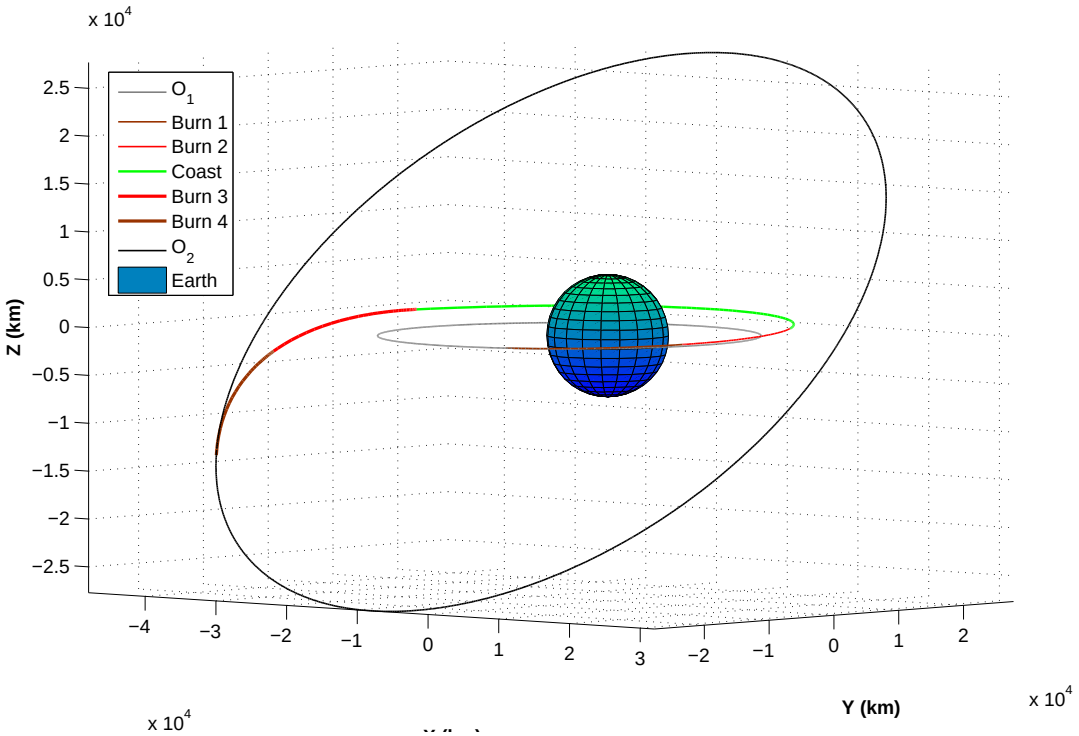


Figure 4.6: Five-Segment Minimum-Fuel Trajectory

4.3 Analysis and Comparison of Numerical Derivatives

The fact that using gradients calculated with complex-step derivatives results in the convergence to the same solution as gradients calculated using other established numerical differentiation methods is useful as far as the implication that these derivatives are a viable method of calculating gradients for a certain class of problems. What the result does not say is how well they compare to central differences or analytical derivatives. Here, the accuracy of

complex-step derivatives is compared to central differences for the impulsive solution to the Orbit Transfer Problem in Section 4.3.1. The robustness, in terms of iterations to converge, is examined for all three methods in 4.3.2. Also, the computational cost, in terms of runtime for the gradient calculation and for the overall optimization, is tested in Section 4.3.3.

4.3.1 Accuracy

As stated in Section 4.2.1, the gradients of the equality constraints, $\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}$, are computed via central differences, complex-step differentiation and analytical derivatives as a function of the State Transition Matrix. The central differences used in this solution are user-input according to

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i + h) - f(x_i - h)}{2h} \tag{4.25}$$

where $h = \epsilon\,(1 + |x_i|)$, and $\epsilon = 1.4 \times 10^{-5}$. This particular ϵ has been determined by another student (described via personal communication with Ricardo Restrepo, February-March 2011) using a method described in [19] to provide the minimum error, balancing the round-off and truncation error, for this problem (specifically, the impulsive solution of this problem). Thus, it can be said that the central differences used in this comparison are more accurate central differences for this problem given a constant ϵ , than an arbitrary choice of the perturbation length may result.

The complex-step derivatives are calculated using a perturbation length

of $1.0i \times 10^{-14}$. This is chosen to allow multiple significant digits to display in MATLAB's long format (16 digits) and to be below the integration tolerances set in *ode45* (1.0×10^{-12}) .

The comparison is based on the analytical derivatives calculation using the Linear Perturbation Analysis method in Section 4.2.1.2. Since these derivatives are accurate to the numerical integration error in determing the State Transition Matrices, they can be used to determine the accuracy of both the central differences and the complex-step derivatives.

This is done by finding the magnitude of the difference for each element of the gradient matrix for central differences and complex-step derivatives respectively from the analytical gradient

$$\Delta \mathbf{c}_{\mathbf{X}_{P,method}} = \Delta \frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\bigg|_{method} = \frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\bigg|_{method} - \frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\bigg|_{STM}. \quad (4.26)$$

A problem here, however, lies in the fact that this gradient is often poorly scaled with elements of enormously varying magnitudes. For example, the central difference matrix and the analytical derivative matrix may be identical to 10 significant digits in two elements, yet the difference could be of the order 10^{-2} in one element and 10^{-12} in another, or perhaps even more extreme.

To atone for this, in addition to the above comparison, the difference matrices found by from Equation 4.26 can be scaled; each element of the difference matrix is divided by the appropriate element in the analytical gradient

matrix

$$\Delta\bar{\mathbf{c}}_{\mathbf{X}_{P,method}} = \frac{\Delta \left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{method}}{\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{STM}} = \frac{\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{method} - \left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{STM}}{\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{STM}}. \tag{4.27}$$

Thus, Equation 4.26 represents a sort of absolute error, while Equation 4.27 gives a sort of relative error. For both comparisons, it is easier to compare scalars than matrices, so the 2-norm of the difference and scaled difference matrix is computed to provide a "magnitude" for each matrix.

As a point of reference, $\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{STM}$ at the first iteration is given in Table 4.3 (to five significant digits) along with what each element represents.

Table 4.3: $\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right|_{STM}$ at Iteration One.

$\left.\frac{\partial \mathbf{c}(\mathbf{X}_P)}{\partial \mathbf{X}_P}\right\vert$	$h_{x_f} - h^*_{x_f}$	$h_{y_f} - h^*_{y_f}$	$h_{z_f} - h^*_{y_f}$	$e_{x_f} - e^*_{xf}$	$e_{y_f} - e^*_{yf}$
t_1	0	0	-2.2979	-3.9293×10^{-5}	4.2887×10^{-6}
Δv_{1_x}	0	0	-5.7588×10^{4}	0.2352	-1.0363
Δv_{1_y}	0	0	7.0872×10^{4}	0.0208	1.0307
Δv_{1_z}	1.8874×10^{4}	-1.1314×10^{4}	0	0	0
t_2	0	0	1.4377	-7.1449×10^{-6}	-2.3837×10^{-5}
Δv_{2_x}	0	0	1.2281×10^{4}	-0.1128	-0.4489
Δv_{2_y}	0	0	-5.4919×10^{4}	0.9808	-0.1238
Δv_{2_z}	-1.2281×10^{4}	5.4919×10^{4}	0	0	0

Values for $\|\Delta\mathbf{c}_{\mathbf{X}_P}\|$ and $\|\Delta\bar{\mathbf{c}}_{\mathbf{X}_P}\|$ for each method are computed at the first and fifth iteration. These values are found in Table 4.4 along with $\Delta\mathbf{c}_{\mathbf{X}_{P,max}}$ and $\Delta\bar{\mathbf{c}}_{\mathbf{X}_{P.max}}$.

Table 4.4: Comparison between Central Differences and Complex-Step Derivatives

Value	$\|\Delta\mathbf{c}_{\mathbf{X}_P}\|_{Iter1}$	$\|\Delta\mathbf{c}_{\mathbf{X}_P}\|_{Iter5}$	$\Delta\mathbf{c}_{\mathbf{X}_{P,max}}\big\|_{Iter1}$	$\Delta\mathbf{c}_{\mathbf{X}_{P,max}}\big\|_{Iter5}$
$\Delta\mathbf{c}_{\mathbf{X}_{P,CD}}$	2.1322×10^{-5}	1.3074×10^{-3}	1.8086×10^{-5}	9.8021×10^{-4}
$\Delta\mathbf{c}_{\mathbf{X}_{P,CS}}$	1.2966×10^{-5}	6.4119×10^{-5}	1.5645×10^{-6}	3.8071×10^{-5}
$\Delta\bar{\mathbf{c}}_{\mathbf{X}_{P,CD}}$	2.6897×10^{-5}	1.4910×10^{-6}	2.6725×10^{-5}	9.0096×10^{-7}
$\Delta\bar{\mathbf{c}}_{\mathbf{X}_{P,CS}}$	1.3287×10^{-8}	2.4910×10^{-8}	1.2872×10^{-8}	1.3800×10^{-8}

It is easily seen that in every possible interest at this point that the complex-step derivatives are significantly more accurate than the central differences.

4.3.2 Robustness

Now that it has been shown that complex-step derivatives outperform even "optimal" central differences in terms of accuracy related to analytical derivatives, it needs to be shown the effect that this has on the ability of the optimizer to converge to the accepted solution. This section describes the ease of convergence of the problem using complex-step derivatives compared to central differences and STM-based derivatives for the impulsive problem and compared to central differences only for both finite-burn problems. In theory, for greater accuracy of the gradients used in the optimization process the optimizer should converge sooner and for a wider range of initial guesses (i.e. the optimization is more robust). In this process, robustness is characterized by two parameters: convergence to accepted solution and number of iterations to converge.

In the optimization of the impulsive-maneuver problem, all three differentiation methods converge to the same solution to known significant digits. Also, both complex-step differentiation and central differences converge to the same solution for both the five-segment and three-segment finite burn solutions. That leaves simply the number of iterations to compare. This is given by Table 4.5 for all solutions.

Table 4.5: Number of Iterations to Converge

Solution	Impulsive	3-Segment	5-Segment
Analytical	54	N/A	N/A
Central Diff.	57	101	167
Complex-Step	54	87	137

It is easy to note the improvement in number of iterations to convergence, and thus robustness, for complex-step derivatives over central differences in every example. Table 4.5 also seems to indicate that an increase in the complexity of the problem, the greater effect the complex-step derivatives have in finding the optimal solution. In the impulsvie case, a three iteration improvement is present compared to 14 iterations for the three-segment case and 30 iterations for the five-segment case.

4.3.3 Computational Cost

In modern computer science there are two ways to define computational cost: function evaluations and runtime. Here, runtime is the primary interest because it seems to be a more direct way to measure this cost. There are two

different runtimes that will be of concern. One is the total optimization time which is defined as the time required for the optimizer to converge once it has been entered with an initial guess. The second is differentiation time; simply the length of time it takes to compute the gradient of interest. Differentiation time indicates the efficiency of calculating a gradient using a particular method. The total optimization time not only takes into account differentiation time, but also how the optimizer takes advantage of more accurate gradients to acheive convergence faster.

Total optimization time is calculated by noting the difference in computer time between the algorithm's entrance into the optimization process and it's exit. This is done using MATLAB's *tic* and *toc* functions. These functions are also used in the calculation of the differentiation time. Each time the function to compute the value of the constraints and the gradients of the constraints is called the differentiation time is defined by the time it takes from the initialization of the gradient matrix until the matrix is filled. This value is output to file where it is stored for later calculation of the mean, maximum, and minimum differentiation times.

Table 4.6 shows the mean, maximum, and minimum differentiation time for each differentiation method as well as the total optimization time for the impulsive solution.

Table 4.6: Impulsive-Solution Time Comparison (all times in seconds)

Method	Analytical	Central Diff.	Complex-Step
t_{OPT}	27.5262	132.6227	62.6643
$t_{deriv}\|_{mean}$	3.4786×10^{-4}	1.2237	0.4779
$t_{deriv}\|_{max}$	0.0085	2.5255	0.5748
$t_{deriv}\|_{min}$	2.000×10^{-4}	0.8025	0.3989

The times in this table show that not only does complex-step differentiation hold an accuracy improvement over central differences but also has a significantly shorter runtime to calculate each derivative. This is primarily a result of the single perturbation step required at to calculate each derivative whereas central differencing requires two perturbation steps for each derivative. Now, this is the case in a MATLAB environment, in other environments that do not intrinsically handle complex numbers, the additional differential equations and conversions necessary may take a large chunk out of this advantage, if not turn it into a slight disadvantage. In addition, the total optimization time is significantly decreased. Besides the factor of the decreased individual differentiation times, the fewer necessary iterations needed provide an additional advantage.

Similar results hold for the three and five-segment solutions as well. For the three-segment, finite-burn case, the results for the various differentiation times along with the total optimization times for each differentiation method are found in Table 4.7 .

Table 4.7: Three-Segment Finite-Burn Time Comparison (all times in seconds)

Method	Central Diff.	Complex-Step
t_{OPT}	230.266	207.767
$t_{deriv}\|_{mean}$	1.4717	0.8884
$t_{deriv}\|_{max}$	1.5276	2.5819
$t_{deriv}\|_{min}$	1.4095	0.8112

For the five-segment, finite-burn case, the results for total optimization time as well as mean, maximum, and minimum differentiation times for both central differences and complex-step derivatives are found in Table 4.8.

Table 4.8: Five-Segment Finite-Burn Time Comparison (all times in seconds)

Method	Central Diff.	Complex-Step
t_{OPT}	1034.089	460.388
$t_{deriv}\|_{mean}$	2.4942	1.4646
$t_{deriv}\|_{max}$	4.1284	2.4910
$t_{deriv}\|_{min}$	2.3904	1.3996

The results presented here strongly indicate a significant advantage in the runtime for complex-step differentiation over central differences as well as in the time to convergence for both methods.

Chapter 5

Summary and Conclusion

Complex-step differentiation is analyzed as an alternative to traditional numerical differentiation methods. Traditional methods are examined, focusing on their advantages and disadvantages. A mathematical basis for complex-step differentiation is presented providing insight to the limitations and advantages of the method. Then, complex-step differentiation is implemented in the lunar ascent problem testing its applicability in linear, quadratic, and cubic interpolation schemes as well as in the use of fixed-step and variable-step integration. Finally, the optimal orbit transfer problem is examined comparing complex-step derivatives to central differences in terms of accuracy, optimization convergence, and computational cost.

5.1 Conclusion

This study shows that complex-step differentiation is an applicable numerical differentiation method for a wide range of tools used in the solution of optimization problems. Complex-step derivatives are demonstrably more accurate than traditional differencing methods. The accuracy is shown to improve

the robustness of given algorithms as well as the computational cost as a result of fewer iterations necessary for convergence. In addition, these derivatives consistently evaluate faster than central differences in a MATLAB environment. The restrictions on this method, that it is limited to analytic functions and requires complex arithmetic, appear to be significantly outweighed by their benefits over traditional methods. In short, complex-step derivatives truly are a fast, accurate and easy to implement numerical differentiation method.

5.2 Recommendations for Future Work

There are two primary directions for future extensions of this work. One, is the extension of testing beyond the MATLAB environment. Implementation of complex-step differentiation into compiled languages such as FORTRAN and C++ is a major step in the use of complex-step derivatives in optimization problems due to the additional speed gained in these environments compared to MATLAB. This implementation is more difficult since these languages do not have the inherent flexibility of MATLAB in handling complex arithmetic, obviously a necessary component for these derivatives. In particular, it is interesting to consider the effect in runtime comparisons between complex-step derivatives and central differences in an environment where the real and imaginary components of the differential equations would need to be separated - essentially doubling the number of integrations performed.

Secondly, an implementation of the work found in [6], an extension of

complex-step derivatives into higher dimensions for use in higher-order derivatives, into a similar study as to this would provide interesting results. In particular, analyzing 2nd order derivatives evaluated through this extension and their use in 2nd order optimization methods and in computing Hessians used in nonlinear programming solvers could provide even more benefits than simple complex-step differentiation.

Bibliography

[1] Lyness, J. N., and Moler, C. B. 1967. "Numerical Differentiation of Analytic Functions." *SIAM Journal on Numerical Analysis.* Vol.4, June 1967, pp. 202-210.

[2] Squire, W. and Trapp, G. 1998. "Using Complex Variables to Estimate Derivatives of Real Functions." *SIAM Review.* Vol. 40, No.1, 1998, pp. 110-112.

[3] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J. 2003. "The Complex-Step Derivative Approximation." *ACM Trans. Math. Softw.* Vol. 29, No. 3, 2003, pp. 245–262.

[4] Lai, K. L. 2006. *Generalizations of the Complex-Step Derivative Approximation.* PhD thesis. University of Buffalo, Buffalo, NY, Sept. 2006.

[5] Shampine, L. F. 2007. "Accurate Numerical Derivatives in MATLAB." *ACM Trans. Math. Softw.*, 33, 4, Article 26 (August 2007), 17 pages.

[6] Lantoine, G., Russell, R. P., Dargent, T. 2010. "Using Multicomplex Variables for Automatic Computation of High-Order Derivatives." Paper AAS 10-218, Feb. 2010. AAS/AIAA Space Flight Mechanics Meeting, San Diego, CA.

[7] Arora, N., Russell, R. P., and Vuduc, R. W. 2009. "Fast Sensitivity Computations for Trajectory Optimization." AAS/AIAA Astrodynamics Specialist Conference and Exhibit, 2009.

[8] Weisstein, Eric W. "Numerical Differentiation." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/NumericalDifferentiation.html Accessed 9 April, 2011.

[9] *Numerical Recipes in C. The Art of Scientific Computing, 2nd Edition*, 1992.

[10] Griewank, A. 1989. "On Automatic Differentiation." *In Mathematical Programming: Recent Developments and Applications.* Kluwer Academic Publishers. 1989. pp. 83-108.

[11] Dixon, L. C. W. 1991. "On the Impact of Automatic Differentiation on the Relative Performance of Parallel Truncated Newton and Variable Metric Algorithms." *SIAM Journal of Optimization.* Vol. 1, No. 4, November 1991, pp. 475-486.

[12] Martins, J. R., Sturdza, P., and Alonso, J. J. 2001. "The Connection Between the Complex-Step Derivative Approximation and Algorithmic Differentiation." AIAA Paper 2001-0921, AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV. Jan. 2001.

[13] Hull, D. G. 2010. "Optimal Guidance for 3D Lunar Ascent." 20th

AAS/AIAA Space Flight Mechanics Meeting, San Diego, CA. 14-17 February 2010.

[14] Cheney, W. and Kincaid, D. 2004. *Numerical Mathematics and Computing, Fifth Edition.* Brooks/Cole-Thomson Learning, Belmont, CA. 2004.

[15] Bate, R.R., Mueller, D. D., and White, J.E. 1971. *Fundamentals of Astrodynamics.* Dover Publications Inc., New York, NY. 1971.

[16] Curtis, H.D. 2005. *Orbital Mechanics for Engineering Students.* Elsevier Butterworth-Heinemann, Burlington, MA. 2005.

[17] Battin, R.H. 1999. *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition.* American Institute of Aeronautics and Astronautics. 1999.

[18] Ocampo, C. and Munoz, J. P. "Variational Equations for a Generalized Spacecraft Trajectory Model." *Journal of Guidance, Control, and Dynamics.* Vol. 33, No. 5, Sept-Oct. 2010.

[19] Ocampo, C. and Hernandez, S. 2011. "Automation of Optimal Control Finite Burn Trajectories." 21st AAS/AIAA Space Flight Mechanics Meeting, New Orleans, LA. 13-17 February 2011.

Vita

Alan Robert Campbell was born in Erie, Pennsylvania on 4 September 1986, the son of Roy A. Campbell and Catherine A. Campbell. He received the Bachelor of Science degree with Honors in Aerospace Engineering from the Schreyer Honors College at the Pennsylvania State University in May, 2009. Following the completion of his undergraduate study, he was accepted into the Graduate School at the University of Texas at Austin and began study of Orbital Mechanics and Controls in the Department of Aerospace Engineering and Engineering Mechanics in August, 2009.

Permanent address: 19778 Morris Rd.

Meadville, Pennsylvania 16335

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.