The Dissertation Committee for Ravishankar Mathur
certifies that this is the approved version of the following dissertation:

# An Analytical Approach to Computing Step Sizes for Finite-Difference Derivatives

Committee:

Cesar A Ocampo, Supervisor

David G Hull

Wallace T Fowler

Belinda Marchand

Juan Senent

# An Analytical Approach to Computing Step Sizes for Finite-Difference Derivatives

**by**

**Ravishankar Mathur, B.S.A.A.E.; M.S.E.**

**DISSERTATION**

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

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN
May 2012

Dedicated to my wife Tajel; may she never have to go through something like this again.

# Acknowledgments

To Dr. Ocampo and Dr. Howell: you gave me second chances, and your teachings sparked and continuously reaffirmed my love for theory and analysis. Dr. Ocampo, you unquestioningly supported me through my search for a dissertation topic about which I could be passionate, and I cannot thank you enough for that. I hope to nurture independent creative thought in others, the way you have in me.

To my wife Tajel: you met me when graduate school was taking over my life, and I can't tell you how much your unending love and support has helped me get through it. You've taught me the definition of the word 'patience', and I hope I can eventually understand it as well as you do.

To my mother and father Jyoti and Aditya: you taught me the discipline required to complete this degree. To have parents who truly understand what one is going through is invaluable, and for that I am thankful every day.

To J.P. Munoz: you were the first friend of mine accomplish the trifecta of a wedding, a Ph.D., and a new career, all within one year. You taught me by example how to handle that without going crazy. Or at least, completely crazy.

Finally, to George Davis and Everett Cary of Emergent Space Technologies: thank you for being so understanding and supportive of my dissertation time requirements.

# An Analytical Approach to Computing Step Sizes for Finite-Difference Derivatives

Publication No. _____

Ravishankar Mathur, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Cesar A Ocampo

Finite-difference methods for computing the derivative of a function with respect to an independent variable require knowledge of the perturbation step size for that variable. Although rules of thumb exist for determining the magnitude of the step size, their effectiveness diminishes for complicated functions or when numerically solving difficult optimization problems.

This dissertation investigates the problem of determining the step size that minimizes the total error associated with finite-difference derivative approximations. The total error is defined as the sum of errors from numerical sources (roundoff error) and mathematical approximations (truncation error). Several finite-difference approximations are considered, and expressions are derived for the errors associated with each approximation. Analysis of these errors leads to an algorithm that determines the optimal perturbation step size that minimizes the total error.

A benefit of this algorithm is that the computed optimal step size, when used with neighboring values of the independent variable, results in approximately the same magnitude of error in the derivative. This allows the same step size to be used for several successive iterations of the independent variable in an optimization loop. A range of independent variable values for which the optimal step size can safely remain constant is also computed.

In addition to roundoff and truncation errors within the finite-difference method, numerical errors within the actual function implementation are also considered. It is shown that the optimal step size can be used to compute an upper bound for these condition errors, without any prior knowledge of the function implementation. Knowledge of a function's condition error is of great assistance during the debugging stages of simulation design.

Although the fundamental analysis assumes a scalar function of a scalar independent variable, it is later extended to the general case of a vector function of a vector independent variable. Several numerical examples are shown, ranging from simple polynomial and trigonometric functions to complex trajectory optimization problems. In each example, the step size is computed using the algorithm developed herein, a rule-of-thumb method, and an alternative statistical algorithm, and the resulting finite-difference derivatives are compared to the true derivative where available.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Accurately computing the derivative of a function is a problem of interest within every engineering discipline. In aerospace subfields, the derivative of a function is most influential in problems for which the solution is obtained via gradient-based nonlinear optimization. In these optimization methods, a function is extremized by analyzing it at a given set of optimization parameters, and then iteratively changing those parameters according to the derivatives of the function. It is reasonable to infer, then, that the outcome of such optimization methods would depend on the accuracy of the computed derivatives.

The following simple minimization problem gives evidence of this dependence: given a point on the $x$-$y$ plane, minimize the point's $y$ coordinate with the constraint that the point should lie on a unit circle. Specifically, the problem is to minimize the performance index

$$J(x, y) = y$$

subject to the equality constraint

$$x^2 + y^2 = 1$$

The analytical solution to this problem is easily shown to be $(x, y) = (0, -1)$.

Table 1.1: Effects of correct vs error-prone derivatives on optimization.

| Derivatives | $x_f$ | $y_f$ | iterations |
|---|---|---|---|
| Exact | $-1.09 \cdot 10^{-11}$ | $-1.00$ | 18 |
| Erroneous | $-2.48 \cdot 10^{-7}$ | $-1.00$ | 43 |

The numerical solutions[1] using correct and error-prone derivatives are given in Table 1.1. Although the optimizer arrives at the same solution in both cases, the iterative path taken differs noticeably when errors are introduced in the computed derivatives. Not only does the optimizer take considerably more iterations to converge, but the convergence tolerance for the $x$ coordinate is not nearly as accurate. In more complex optimization problems, such as those arising from real-life scenarios, a change in the iterative path could lead to a different solution or even to no converged solution at all.

Because of the fundamental importance of derivatives within engineering, an enormous amount of attention has been given to the general problem of efficiently computing accurate derivatives. Solutions have ranged from simple algorithms that work for any problem but are error-prone, to highly specialized complex algorithms with near-analytical accuracy. The finite-difference class of algorithms are among the simpler algorithms, both mathematically and in terms of implementation complexity. On the opposite end of the difficulty spectrum are problem-specific solutions such as the Variational Model by Ocampo et al. [33–35],

[1]Numerical solutions were obtained using the VF13 SQP algorithm, which is part of the Harwell Subroutine Library [47].

which finds near-analytical partial derivatives for spacecraft trajectories involving one or more segments.

Every finite-difference algorithm contains a parameter, called the step size, which controls the accuracy of the computed derivative. If the chosen step size is too large, then mathematical truncation error dominates the derivative. If it is too small, then numerical roundoff errors greatly reduce the derivative's accuracy. To understand the impact of the step size, consider the process of optimizing a three-impulse transfer from a Moon-centered initial orbit to a specified $\mathbf{V}_\infty$ vector, as described in Whitley et al. [56]. Assuming a circular initial lunar orbit, the general geometry of such a transfer is as follows. The first impulse is mostly an apoapse-raising maneuver, the second impulse (near apoapse) mainly changes plane, and the third impulse (near periapse) inserts the spacecraft onto a departure hyperbola with the desired $\mathbf{V}_\infty$ vector. Although it has been shown by Gobetz and Doll [14] that such a maneuver often outperforms a single-impulse transfer, the added complexity of coordinating multiple impulses makes numerical optimization a difficult task. This difficulty occurs, in no small part, because computing derivatives for the system Jacobian matrix using finite-difference methods requires considerable effort to find suitable step sizes. Ocampo and Saudemont [36] and Jones and Ocampo [21] have made great strides in efficiently computing initial guess solutions for the three-impulse transfer, which helps to increase the chance of successful optimization. However, due to the sheer amount of time spent finding step sizes (especially after the impulses were converted to finite burns), Ocampo et al. [33, 35] developed an alternate specialized method just to compute the deriva-

tives of the problem.

Situations like the one described, where researchers put much effort into developing alternatives to finite-difference derivatives, are not at all uncommon. When one spends countless hours fine-tuning the step sizes to achieve convergence in one set of test cases, only to find that another set of test cases no longer converges, it is understandable to seek out an alternative method to compute derivatives.

This dissertation focuses on the problem of finding an optimal balance for the step size, known as 'the step-size dilemma'. Formally, the problem is to find the step-size value that minimizes both mathematical and numerical errors for a specified finite-difference method, given a function and particular values of the function's domain variables.

## 1.1 Existing Work on Numerical Derivatives

### 1.1.1 Finite-Difference Derivative Approximations

Finite-difference approximations are arguably the most frequently studied methods for computing the derivative of a function. They are derived from Taylor's Theorem [52][2], written in 1715, which approximates a function near a point by a polynomial of that function's derivatives [20]. As such, finite-difference methods essentially fit a polynomial to a function at a given point, and then estimate the function's derivative to be the polynomial's derivative at that point.

[2] An excellent modern essay on Taylor's Theorem is given by Gibson [11].

It is expected that a polynomial fit to a general function will have errors, and from Taylor's Theorem, these errors take the form of an infinite series[3]. In 1797, Lagrange was among the first to reduce this infinite series to a succinct finite term in his landmark work, *Théorie des Fonctions Analytiques* [24]. The Lagrange remainder of the Taylor Series of $f(x)$ about a point $x_0$ truncated after $n$ terms is

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \tag{1.1}$$

where $\xi \in (x_0, x)$ is the only unknown. By replacing an infinite series representation with a single unknown variable $\xi$, Lagrange opened the door to modern numerical step-size analysis.

In the early 1900's, Richardson and Gaunt [43, 44] introduced and refined the Richardson Extrapolation method. This method produces a high-order approximation to a function by combining several low-order approximations. It has been used successfully in fluid dynamics (for estimating fine meshes by using several coarse meshes) and numerical integration methods [4, 5]. When applied to finite-difference derivatives, Richardson Extrapolation provides a high-order derivative estimate using low-order estimates with successively decreasing step sizes.

By the 1970's, a considerable amount of headway had been made on the topic of estimating an optimal step size. Curtis and Reid [6] described a simple method which requires a priori estimation of higher-order derivatives and roundoff

[3]While Taylor originally wrote the error term as a multiple integral, it is now commonly represented as an infinite summation series.

errors of the function. While the resulting step sizes give decent results for many problems, estimating the function's roundoff errors is no easy task for non-trivial functions (a fact even more true today, as functions get extremely complex).

In 1979, Stepleman and Winarsky [51] created an adaptive step-size algorithm which iteratively evaluated a descending sequence of step sizes. Using the fact that the finite-difference derivative monotonically tends to the true derivative (for a 'sufficiently small' sequence of step sizes), they showed that the smallest step size for which the finite-difference derivative did NOT violate this trend is in fact the optimal step size. Because of the caveat that the initial step size in the search sequence must be sufficiently small, they developed a heuristics-based method for estimating the initial step size. Interestingly, some of the core methods of this dissertation are similar to those of Stepleman and Winarsky, and these similarities are compared and contrasted herein.

In 1983, Gill, Murray, Saunders, and Wright [12] presented an algorithm to compute step-sizes for forward-difference derivative approximations (an extension to Gill et al. [13] in 1981). Not happy with the significant number of function evaluations required by iterative methods, their method instead gave a succinct equation by which an 'optimal' step size could be computed for forward- and central-difference derivatives. It required an estimate of the function's condition error (which can be estimated by a method given by Hamming [19]), and explicitly ignored subtractive cancellation error.

In contrast to the previously common assumption that a function has many digits of accuracy, in 1992 Barton [1] showed how to approximate the step size

when the function of interest had only a few reliable digits of accuracy. In addition, Barton argued that when finite-difference derivatives are used as part of an optimization loop, it is important to recompute the step sizes as the optimization progresses. He also offered a rudimentary method by which it can be determined how often to recompute step sizes (relative to a change in the optimization variable). Barton did, in fact, rely heavily on results presented by Gill et al. [12]; his main contributions were in the treatment of finite-difference step sizes within the framework of optimization problems and error-prone functions.

Yang et al. [58] give a practical approach to computing finite-difference step sizes in their 2005 book. Referring to the fact that the majority of step-size estimation methods require a priori knowledge of roundoff errors within the function of interest, they conclude that

> ... these equations are only of theoretical value and cannot be used practically to determine $h_0$ [the step size] ...

Yang is among the first to acknowledge the fact that such detailed knowledge of a function is often not available.

Diverging from the trend of computing an optimal step size, in 2011 Prentice [38] cleverly improved upon the standard 3-point approximation (commonly called central differences) by computing an optimal set of sampling points. Instead of evaluating the function at $f(x-h)$ and $f(x+h)$, the method computes a set $\{a, b, c\}$ which are used to compute $f(x+ah)$, $f(x+bh)$, and $f(x+ch)$. These

non-symmetric function values, when used with a standard 3-point method, were shown to actually *increase* the order of the method beyond the standard $O(h^2)$.

Although much research was being done in the latter half of the $19^{th}$ century on determining good step sizes, it was widely recognized that *any* finite-difference method has a significant loss of accuracy due simply to the finite-precision subtractions involved. In order to avoid this problem altogether, researchs in the numerical analysis community were seeking out alternatives to finite-difference methods. The two main approaches developed are called Automatic Differentiation (AD) and complex-step differentiation (CSD), and it is important to know their histories in order to compare and contrast them to finite-difference methods.

### 1.1.2 Complex-Step Derivative Approximations

Early forays into alternate numerical differentiation schemes were by Lyness and Moler [28, 29] in the late 1960's. By noticing that the Cauchy integral theorem gives the $n^{th}$ derivative of a complex-domain function as a closed complex integral, they were able to formulate methods to numerically compute derivatives. This was done by evaluating the complex integral using trapezoidal quadrature methods and choosing a particular contour in the complex plane.

Fornberg [9] extended this work in 1981 by using the Fast Fourier Transform to evaluate the complex integral. His work aimed to implement the same theoretical solution as Lyness and Moler using a more robust and compact algorithm that required less user interaction.

For nearly two decades the field of complex analysis in numerical differenti-

ation remained dormant, until Squire and Trapp [50] introduced the complex-step method in 1998. This method is not based on the Couchy integral theorem, but instead simply uses complex perturbations with the general Taylor series. The resulting well-known formula,

$$f'(x) = \frac{Im[f(x+ih)]}{h} \tag{1.2}$$

is considerably easier to implement than the methods of Lyness and Moler and Fornberg, and has no subtractive cancellation error as do finite-difference methods.

While Squire and Trapp sought only to state the complex-step formula in (1.2) and analyze numerical results, a thorough and in-depth analysis of the CSD method was performed by Martins, Sturdza, Alonzo, and Kroo [30–32] between 2000 and 2003. In addition to analyzing the CSD method, their papers explored its relationship to AD methods and provided computer scripts to automatically make existing functions compatible with the CSD method.

Following the papers of Martins et al., the scientific community picked up quickly on this simple new method of computing derivatives to almost analytical precision, and many developers have turned to its use for numerical differentiation. However, because the CSD method was designed to compute only the first derivative, several researchers (including Lai [25]) sought to extend the theory to higher-order derivatives only to discover that the results were just as prone to cancellation errors as finite-differences.

In 2006, Pemba [37] developed an extension to CSD which allowed for any order derivative to be computed without any finite-difference cancellation errors.

Their method used a complexification function to iteratively compute higher-order derivatives from lower-order ones.

It was not until 2010 that a true generalization of the CSD method was introduced by Lantoine, Russell, and Dargent [26, 27]. Their method used multicomplex numbers in place of ordinary complex numbers, which allowed them to derive an arbitrary-order derivative by perturbing only the appropriate multicomplex direction of the independent variable. The resulting derivatives are not subject to any finite-difference cancellation errors, and can therefore be computed to an arbitrary precision.

### 1.1.3 Automatic Differentiation

When an equation or algorithm is implemented as a computer program, it is invariably broken down into a series of steps, each one involving an elementary operation. Automatic Differentiation methods take the partial derivatives of each of these operations, and then combine them via the chain rule to produce the desired derivative. The early days of AD (the 1960's and 1970's) involved specialized precompilers, often times with their own specialized languages. [40, 41, 54, 55]

Instead of developing a specialized AD language, Kedem [23] considered the differentiation of a Fortran subroutine. His 1980 paper suggested using an automated process to analyze an existing Fortran subroutine and replace elementary operations by their derivatives. This method forms the basis of AD methods today.

## 1.2 Motivation and Research Contributions

There is no doubt that, when used properly, the complex-step and automatic differentiation methods outperform finite-difference derivative methods in terms of accuracy. The question then arises: Why continue studying methods to find optimal step sizes for finite-difference derivatives? The iterative step-size search method of Stepleman and Winarsky [51] could suffice where necessary, and all future research could be directed towards CSD and AD methods.

To answer this question, one must consider the current climate of numerical differentiation. Finite-difference methods are so easy to implement and verify that their use has become de facto within industry. Because CSD methods did not really take off until the 2000's, and AD methods would have been too slow to implement effectively on computers until the same time, finite-difference methods were also the only feasible choice. Therefore, there are currently uncountably many simulations in use that employ this method. Within many of these simulations, the function to be differentiated consists of calls to libraries and other computational facilities for which the source code is either not available, or is not easy to obtain. Even if this is not the case, and the function is purely self-contained, often times it either cannot be changed (perhaps due to verification requirements) or is very large and therefore difficult to change and verify.

All CSD methods require that the function of interest be able to operate on complex numbers. Some modern programming environments such as MATLAB assume complex numbers intrinsically, at least for built-in capabilities and most

user-built programs[4]. Fortran, C[5], and C++ all support complex variables, however they must be explicitly declared as such. Because the vast majority of existing simulations were created long before the rise of CSD, they do not have variables declared as complex. In order to use the CSD method with these simulations, the function(s) to be differentiated must have all relevant variables converted to the complex data-type, either manually or with existing CSD conversion scripts.

In contrast, AD methods can operate in two basic forms. The first method uses special AD preprocessors to analyze the function, differentiate it instruction-by-instruction, and create a new function that computes the derivatives of the original. The second method requires that the function itself be written using special AD-friendly data types, and uses operator overloading to create code for derivative evaluation at compile time.

For both AD and CSD methods, the entire function of interest must be available and editable at the source code level. This instantly precludes all codes that call external precompiled libraries as part of critical computations. As discussed earlier, these codes are not rare, and in today's world of object-oriented programming it is becoming even more common to use scientific libraries written by third parties.

From the reasons laid out here, it is clear that there still exists a demand for accurate finite-difference derivatives. Since increasing the accuracy of

[4]The exception in MATLAB would be custom specialized algorithms, for which care must be taken to ensure that the algorithm itself can handle complex variables.

[5]C supports complex data types as of the C99 specification.

finite-difference derivatives goes hand-in-hand with choosing a better step size, the search for an optimal step size as discussed in this dissertation is very relevant to the simulation community.

## 1.3 Dissertation Organization

This dissertation introduces and proposes a solution for the finite-difference step-size dilemma in five chapters, including this introduction.

Chapter 2 details the analytical solution by identifying the main sources of error, formulating equations to quantify those errors, and then analyzing those equations to determine the optimal step size. Informal proofs are given to show that the derived approximations to the errors do in fact follow the same trends as the errors themselves. It is shown that most of the analysis can be done knowing only the order of the finite-difference method, without regard to the method's particular equation. For the small part in which the particular finite-difference equation does appear, a variety of common equations are considered and their results are tabulated for convenience. Finally, arguments are made concerning the validity of the optimal step size as the independent variable of interest changes in value (which happens at ever iteration of an optimization loop).

Chapter 3 presents a rigorous approach to developing an algorithm which bridges the gap between the analytical and numerical sides of the step-size dilemma. The algorithm is first developed for the one-dimensional case, and then extended to the multidimensional case while retaining memory efficiency. For the multidimensional case, the algorithm finds several optimal step sizes (one for each element

of the function's output), so methods are discussed for choosing a particular step size. Finally, an implementation of this algorithm, called AutoDX, is presented and briefly explained.

Chapter 4 studies the effectiveness of AutoDX in finding suitable step sizes for various example problems. The initial examples consider simple polynomial and trigonometric functions. Special cases such as low-order polynomials or zero derivatives are also shown, in order to showcase the exception-handling capabilities of the AutoDX algorithm's underlying analytical solution. Next, intermediate examples consider functions whose implementations are advanced algorithms such as numerical integrations or root-finding problems. These examples show cases where the function's implementation may introduce considerable errors into its output. Finally, full numerical optimization problems are tackled, and the results are compared between AutoDX and several other numerical differentiation methods.

Finally, Chapter 5 concludes this dissertation by identifying opportunities for future research on the step-size dilemma. The AutoDX code itself is summarized in Appendix A, and expressions for several common finite-difference derivative methods are given in Appendix B.

# Chapter 2

# Analysis of Finite-Difference Derivatives

## 2.1 Chapter Summary

This chapter focuses on developing the mathematical tools necessary for step-size analysis. All sources of error within finite-difference derivative methods (hereafter referred to as FDD methods) are identified, and equations to quantify them are derived. This allows for the modeling of total error for a given FDD method, expressed as an upper bound. Because this total error estimate is useful only in the theoretical sense, another equation is developed which estimates the true truncation error of a given FDD method by using a variation of Richardson Extrapolation. It is shown that this estimate accurately matches the theoretical total error model, thereby making it a useful tool in the search for the optimal step size.

## 2.2 Richardson Extrapolation

Let $f(x)$ be an unknown function with a computable approximation $FD_n(x, h)$ whose $n^{th}$-order error formula is given by

$$f(x) = FD(x, h) + ah^n + O(h^{n+k}) \tag{2.1}$$

where the step size $h$ is positive, $n$ and $k$ are known integer constants, and $a$ is an unknown constant. If the approximation is evaluated at two different step sizes $h_1$ and $h_2$ (where $h_1 > h_2$[1]), then

$$f(x) = FD(x, h_1) + ah_1^n + O(h_1^{n+k}) \tag{2.2}$$

$$f(x) = FD(x, h_2) + ah_2^n + O(h_2^{n+k}) \tag{2.3}$$

This constitutes a system of two equations in two unknowns ($f(x)$ and $a$). Omitting intermediate algebra, the solution for $f(x)$ is

$$f(x) = \frac{(h_1/h_2)^n FD(x, h_2) - FD(x, h_1)}{(h_1/h_2)^n - 1} + O(h_1^{n+k}) \tag{2.4}$$

where the order of the error has increased to $n+k$. This is the basis of Richardson Extrapolation; approximations of a given order using two different step sizes are combined to achieve an approximation of higher order. Alternately, solving for the unknown constant $a$ gives

$$a = \frac{FD(x, h_2) - FD(x, h_1)}{h_1^n - h_2^n} + O(h_1^k) \tag{2.5}$$

This estimate of the constant $a$ can then be used to estimate the error $ah_1^n$ in the original approximation $FD_n(x, h_1)$.

## 2.3 The Taylor Series and Lagrange Remainder

The Taylor Series is an infinite series that estimates a function in the neighborhood of a particular point by building up a polynomial whose coefficients

[1]It is often assumed that $h_2 = h_1 t$ for some $0 < t < 1$, but this is not strictly necessary from a derivation standpoint.

depend on successive derivatives of the function. For a function $f(x)$, the Taylor Series of $f(x)$ about the point $x_0$ is

$$f(x) = \sum_{m=0}^{\infty} \frac{f^{(m)}(x_0)}{m!}(x - x_0)^m = f(x_0) + f'(x_0)(x - x_0) + \cdots \tag{2.6}$$

The series on the right-hand-side of (2.6) does in fact converge to $f(x)$ if (and only if) the function is analytic at $x_0$ [18]. In addition, if convergence for a given $x_0$ is achieved over an interval $|x - x_0| < \delta$ for some $\delta > 0$, then $\delta$ is called the *radius of convergence*.

For the remainder of this dissertation, it is assumed that the function being differentiated is analytic, and its radius of convergence is large enough to overstep numerical roundoff errors. Note that this is a perfectly reasonable assumption. If the function is not analytic, then its Taylor Series does not converge to itself (for points other than the trivial $x = x_0$), and therefore FDD methods cannot be used in the first place. If the radius of convergence is very small, then proportionally small step sizes must be used to compute the Taylor Series. Such small step sizes may be well within the roundoff error region for the function.

The basic process of FDD methods is to truncate the Taylor Series after a certain number of terms, and consider the discarded terms as having negligible error for sufficiently small step sizes. The ability to quantify this error forms the basis of step-size analysis. Consider the Taylor Series of $f(x)$ about $x_0$ after $n$

terms,

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots + R_n \tag{2.7}$$

$$R_n = \sum_{m=n}^{\infty} \frac{f^{(m)}(x_0)}{m!}(x - x_0)^m \tag{2.8}$$

The remainder term $R_n$ is actually derived from an integral form[2],

$$R_n = \int_{x_0}^{x} \cdots \int_{x_0}^{x} f^{(n)}(x) dx^n \tag{2.9}$$

If it is assumed that the $n^{th}$ derivative of $f$ is bounded over the interval of integration $[x_0, x]$, i.e. $a \leq f^{(n)}(x) \leq c$, then

$$\int_{x_0}^{x} \cdots \int_{x_0}^{x} a(dx)^n \leq R_n \leq \int_{x_0}^{x} \cdots \int_{x_0}^{x} b(dx)^n \tag{2.10}$$

$$a\frac{(x - x_0)^n}{n!} \leq R_n \leq c\frac{(x - x_0)^n}{n!} \tag{2.11}$$

It is clear that the remainder might take on the same form as the left and right sides of (2.11). In other words, there exists some $b \in [a, c]$ for which

$$a\frac{(x - x_0)^n}{n!} \leq b\frac{(x - x_0)^n}{n!} \leq c\frac{(x - x_0)^n}{n!} \tag{2.12}$$

Equating (2.11) and (2.12), the remainder becomes

$$R_n = b\frac{(x - x_0)^n}{n!} \tag{2.13}$$

Although $b$ is unknown, the Intermediate Value Theorem implies that there exists some point $\xi \in [x_0, x]$ for which $f^{(n)}(\xi)$ will take the value $b$.

$$f^{(n)}(\xi) = b \ , \ \xi \in [x_0, x] \tag{2.14}$$

[2]The modern derivation of the Taylor Series, not given here, is done using integral calculus [18].

From this, the remainder term, called the Lagrange remainder, is rewritten as,

$$R_n = \frac{f^{(n)}(\xi)}{n!}(x - x_0)^n \ , \ \xi \in [x_0, x] \tag{2.15}$$

Note that there is no way to exactly determine $\xi$ for a general function. The $n^{th}$-order Taylor Series for an analytic function $f(x)$ about $x_0$, using the Lagrange remainder, is then

$$f(x) = \sum_{m=0}^{n} \frac{f^{(m)}(x_0)}{m!}(x - x_0)^m + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \ , \ \xi \in [x_0, x] \tag{2.16}$$

In the case where $x < x_0$, (2.16) still holds with the modification that $\xi \in [x, x_0]$.

## 2.4 Derivation of Finite-Difference Methods

The Taylor Series from (2.16) is used to derive all FDD methods. The simplest FDD method, called the forward-difference approximation, is computed from the first-order Taylor Series about the point $x$ itself.

$$f(x + h) = f(x) + f'(x)h + \frac{f^{(2)}(\xi)}{2}h^2 \ , \ \xi \in [x, x + h] \tag{2.17}$$

$$f'(x) = \frac{f(x + h) - f(x)}{h} - \frac{f^{(2)}(\xi)}{2}h \tag{2.18}$$

$$FD_1^{(1)}(x, h) = \frac{f(x + h) - f(x)}{h} + O(h) \tag{2.19}$$

Note the distinction between the true derivative $f'(x)$, and the finite-difference derivative $FD_1^{(1)}(x, h)$, namely that the FDD truncates the error term and represents it simply as an order-of-magnitude. The generic notation $FD_n^{(d)}(x, h)$ will be used to refer to the finite-difference approximation of the $d^{th}$ derivative of $f(x)$,

using a Taylor Series of order $p = n + d - 1$ (where $n, d \geq 1$) and step size $h$. The truncation error associated with such a FDD is easily shown to be $O(h^n)$.

Analysis of approximations using higher-order Taylor Series requires further evaluation of the error term. Consider the common second-order central differences method, computed from the second-order Taylor Series,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f^{(3)}(\xi^+)}{3!}h^3 \ , \ \xi^+ \in [x, x+h] \tag{2.20}$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f^{(3)}(\xi^-)}{3!}h^3 \ , \ \xi^- \in [x-h, x] \tag{2.21}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(\xi^+) + f^{(3)}(\xi^-)}{2}\frac{h^2}{3!} \tag{2.22}$$

The error term in (2.22) can be simplified by observing that it contains the average of the third derivative evaluated at the unknown $\xi^+$ and $\xi^-$ values. Assuming that $f^{(3)}(x)$ is smooth and bounded over $[x-h, x+h]$, the Mean Value Theorem can be used to show that there must exist some $\xi$ between $\xi^-$ and $\xi^+$ which satisfies the average (see Figure 2.1). However, because $\xi^-$ and $\xi^+$ can lie anywhere in their respective intervals, $\xi$ will lie somewhere in the combined interval. With this replacement, the FDD equation for the second-order central-difference approximation becomes,

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(\xi)}{3!}h^2 \ , \ \xi \in [x-h, x+h] \tag{2.23}$$

$$FD_2^{(1)}(x,h) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{2.24}$$

Both (2.19) and (2.24) follow the general form of a FDD equation,

$$FD_n^{(d)}(x,h) = \frac{\Delta f_n^{(d)}(x,h)}{h^d} + O(h^n) \tag{2.25}$$

where $\Delta f_n^{(d)}$ is the appropriate finite-difference expression from Appendix B.

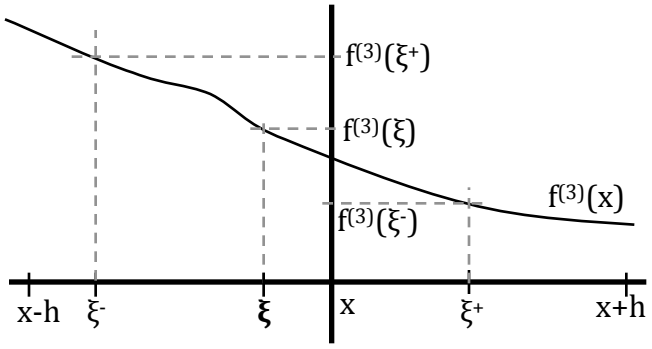


Figure 2.1: The Mean Value Theorem guarantees existence of $\xi \in [\xi^-, \xi^+]$ for which $f^{(3)}(\xi)$ is the average of $f^{(3)}(\xi^+)$ and $f^{(3)}(\xi^-)$.

## 2.5 The Step-Size Dilemma

If the truncation error term in (2.19) is ignored, then it becomes an approximation to the true derivative, with an error proportional to the chosen step size $h$. A cursory examination of the truncation error shows that $O(h) \to 0$ as $h \to 0$, which implies that $h$ should be made as small as possible to maximize the accuracy of the approximation. The same is true of (2.24), in which $O(h^2) \to 0$ even faster than in the forward difference case.

However, a more in-depth analysis of (2.19) and (2.24) indicates a contradiction to the above rule. On any finite-precision machine such as a computer, numbers are represented with a fixed number of binary digits [15]. Because of this, all mathematical operations have an inherent loss of accuracy, as extra digits involved in the computation must be discarded. This phenomenon, called roundoff error, has a significant effect on the subtraction in (2.19) and (2.24). In particular, the relative errors caused by the subtraction operation tend to increase as the step size $h$ is decreased, implying that $h$ should be made as large as possible

to minimize these errors. This contradiction to the preceding requirement is the step-size dilemma.

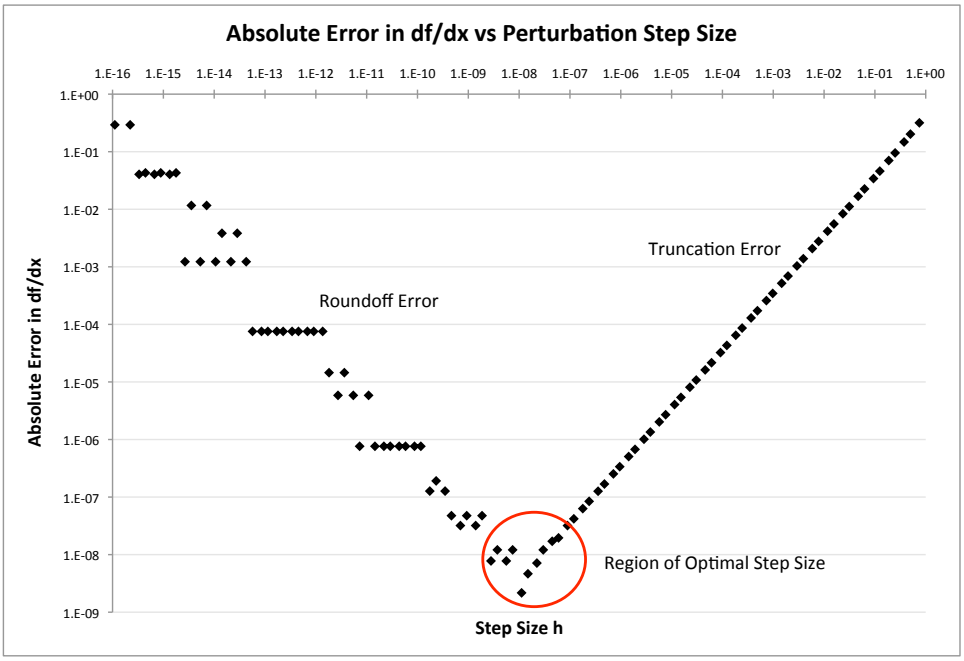


Figure 2.2: The step-size dilemma for $\sin(x)$ with $x = \pi/4$ using the $O(h)$ forward-difference approximation.

The classical step-size dilemma is illustrated in Figure 2.2 by plotting the error in the finite-difference derivative as a function of the step size $h$, using double-log scaling to compress the vast orders of magnitude. It is seen that there exists a region of $h$ for which the total error in $f'(x)$ is minimized. If $h$ is chosen at the low end of that range, then increased roundoff error is sacrificed in favor of decreased truncation error. Conversely, if $h$ is chosen at the high end of that range, then truncation error in the FDD approximation increases, but roundoff error is virtually nonexistent. The optimal choice of $h$ depends on requirements

of the quality of the derivative, as discussed in Chapter 3.

## 2.6 Truncation Error Estimation

Although the value of $\xi$ in the truncation error term of (2.19) is unknown, it is clear that as the step size is reduced, $\xi$ approaches $x$ itself and therefore the remainder term approaches a limit.

$$\lim_{h \to 0} O(h) = -\frac{f^{(2)}(x)}{2} h \tag{2.26}$$

This is the expected truncation error in a forward-difference approximation using a sufficiently small step size $h$, assuming that $f^{(2)}(x)$ is smooth and bounded in the neighborhood of $x$. For the central-difference approximation in (2.24), the same assumptions lead to

$$\lim_{h \to 0} O(h^2) = -\frac{f^{(3)}(x)}{3!} h^2 \tag{2.27}$$

Because each finite-difference approximation has a unique truncation error based on its particular formulation, it becomes necessary to develop a general method to approximate the truncation error. This is done by analyzing the general form of a finite-difference equation, regardless of its order of accuracy. Using (2.19) and (2.24) as a template, a general relationship between a derivative and its FDD approximation is given by,

$$f^{(d)}(x) = FD_n^{(d)}(x, h) + C(x, h)h^n \tag{2.28}$$

where $f^{(d)}(x)$ is the true $d^{th}$ derivative, $FD_n^{(d)}(x, h)$ is the particular finite-difference approximation (in the absence of roundoff errors) given in (2.25), $n$ is the order

of the approximation, and $C(x,h)$ is the coefficient of the truncation error term. In Lagrange form, this is

$$C(x,h) = a_1 f^{(n+d)}(\xi)\ ,\ \xi \in [x - a_2 h, x + a_3 h] \tag{2.29}$$

where $a_1$, $a_2$, and $a_3$ are known nonzero constants determined by the particular finite-difference approximation. Although $C(x,h)$ is undetermined (because it involves the unknown $\xi$), as the step size $h$ is reduced, a similar argument can be used as in the beginning of this section to claim that

$$\lim_{h\to 0} C(x,h) \approx C_n(x) \equiv a_1 f^{(n+d)}(x) \tag{2.30}$$

In other words, if $f^{(n+d)}(x)$ is smooth and bounded in the neighborhood of $x$ and $h$ becomes sufficiently small, the FDD remainder coefficient $C(x,h)$ loses its dependence on the step size. The value of $C_n(x)$ can then be estimated by noting that (2.28) is very similar to the Richardson Extrapolation equation (2.1). Using that method, (2.28) is evaluated with two sufficiently small step sizes $h_1$ and $h_2$ (assuming $h_1 > h_2$),

$$f^{(d)}(x) = FD_n^{(d)}(x,h_1) + C_n(x)h_1^n \tag{2.31}$$

$$f^{(d)}(x) = FD_n^{(d)}(x,h_2) + C_n(x)h_2^n \tag{2.32}$$

Since $C_n(x)$ is considered constant with respect to the step size, (2.5) gives

$$C_n(x) = \frac{FD_n^{(d)}(x,h_2) - FD_n^{(d)}(x,h_1)}{h_1^n - h_2^n} \tag{2.33}$$

It is important to understand the context of (2.33). First and foremost, it gives an *approximation* to the true coefficient $C(x,h)$. In addition, although it assumes

that the step sizes $h_1$ and $h_2$ are sufficiently small, (2.33) does not take into account any numerical issues from step sizes that are too small. In later sections, this seemingly obvious oversight will form the basis of detecting when an optimal step size has been found. For now, the computed $C_n$ can be used to estimate the truncation error for the FDD of order $n$ in (2.28),

$$TE_n(x, h_1) = C_n(x)h_1^n \tag{2.34}$$

Note that the notation indicating which derivative $d$ is being computed has been dropped. This is because although $C_n$ does use finite-difference approximations for a particular derivative, it does not explicitly depend on which derivative is being computed. In addition, this estimate can also be applied to the truncation error associated with step size $h_2$, but the decision to use it with $h_1$ arises from the need to obtain an upper bound on the truncation error, since $h_1 > h_2$.

An example of the use of the truncation error estimate from (2.33) and (2.34) is given in Figure 2.3. The similarities between this estimated truncation error and the absolute error from Figure 2.2 are clear. Both exhibit a decreasing truncation error until a limiting point is reached, followed by an increasing round-off error. The region in which truncation error begins to give way to roundoff error contains the optimal step size. Note the 'stray minima' points within the roundoff error portion of Figure 2.3, for which the estimated error is zero. These points occur because successive finite-difference approximations from (2.33) are equal (due to roundoff errors), causing the estimate for the truncation error to be zero. Such stray minima are not uncommon for step sizes that are so small

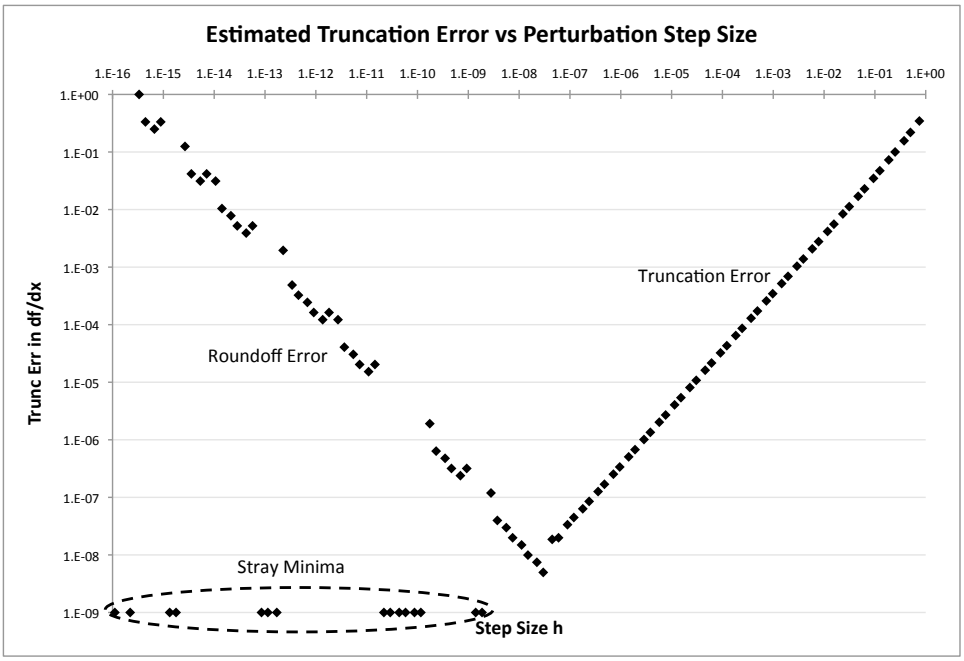


Figure 2.3: Estimated truncation error for $\sin(x)$ with $x = \pi/4$ using the $O(h)$ forward-difference approximation.

that roundoff error dominates the computations. However, their existence does not lessen the effectiveness of using estimated truncation error plots to analyze error trends.

Although Figures 2.2 and 2.3 appear similar, the concern arises as to whether this is merely a coincidence for the particular function being analyzed, or whether an estimated truncation error plot is in fact a good approximation to the true absolute error plot for *any* function[3]. This concern is addressed in later sections of this dissertation, but at this point it is stated that truncation

[3]That is, any function analytic at $x$ for which $f^{(n+d)}$ is smooth and bounded.

error estimates obtained via (2.33) and (2.34) do in fact accurately estimate both truncation and roundoff errors for step sizes that are not too large.

So far, all truncation error analysis has assumed a 'sufficiently small' step size $h$. For the sake of completeness, the opposite case of a large step size should be considered. The general form of a FDD equation, given in (2.28) and (2.29), comes from the Lagrange remainder derivation and therefore does not rely on the step size being small. For a large $h$, (2.29) indicates that the unknown $\xi$ can take a large range of values. Because there is no restriction on the quality of $f^{(n+d)}$ in a large neighborhood of $x$, it is entirely possible that $C(x, h)$ could have a very large magnitude. In addition, for large step sizes, $h^n$ also increases. As a result, using a large step size can result in a truncation error that is large, or worse, unpredictable.

This behavior is seen in Figure 2.4, which shows the difference between estimated and true truncation error in computing the derivative of $f(x) = \sin(x)$ at $x = \pi/4$ using the forward-difference approximation. For step sizes that are too small, the estimate clearly follows the true roundoff error (with some overestimation as discussed in Section 2.9). For step sizes that are 'sufficiently small', the estimate and true errors are almost exactly equal, which is expected since the estimate is designed to ideally model this region. For large step sizes, it is clear that the truncation error estimate is not even close to the true error, which supports the theoretical claim that the truncation error estimate is unreliable for large $h$. The tipping point between invalid and valid truncation errors (in terms of step size) depends highly on the function itself, and to an extent on the point

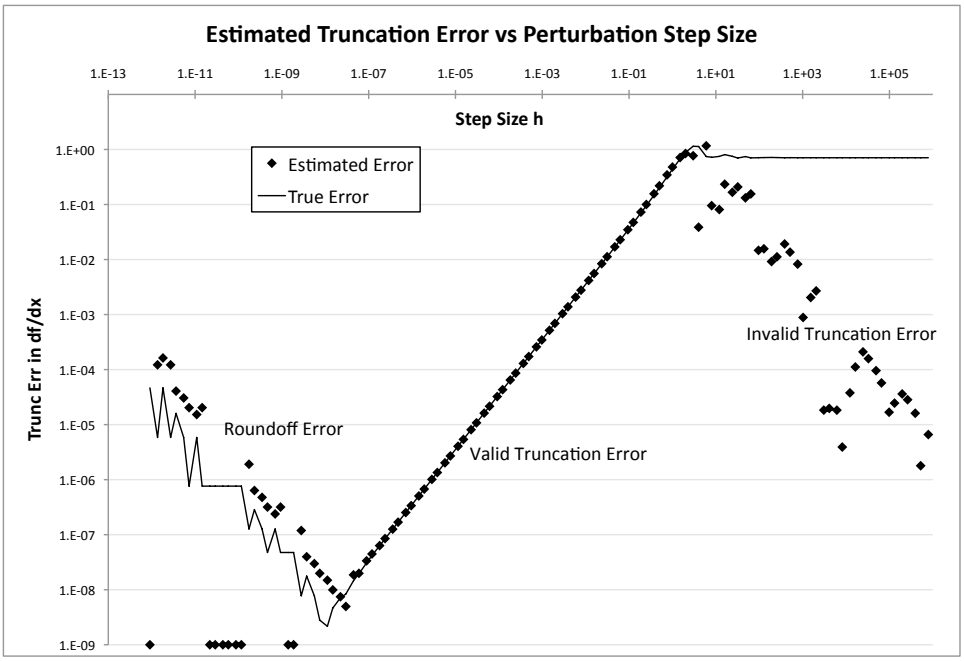


Figure 2.4: Comparison of estimated and true truncation errors for $\sin(x)$, $x = \pi/4$.

$x$ at which the derivative is computed.

A visual analysis of Figure 2.4 indicates that the optimal step size $h_{opt} \approx 10^{-8}$. Given a machine precision of $10^{-16}$ and $x$ having approximately unity magnitude, this result is consistent with conventional approximations [1,12]. However, Figure 2.5 paints a very different picture.

The most significant change between Figures 2.4 and 2.5 is a much smaller valid truncation error interval; 8 orders of magnitude in the former, versus only 5 orders of magnitude in the latter. Once again, the stray points with abnormally small roundoff error are simply fortuitous coincidence. Figure 2.5 also indicates an optimal step size $h_{opt} \approx 10^{-11}$. If the function itself is known – in this case

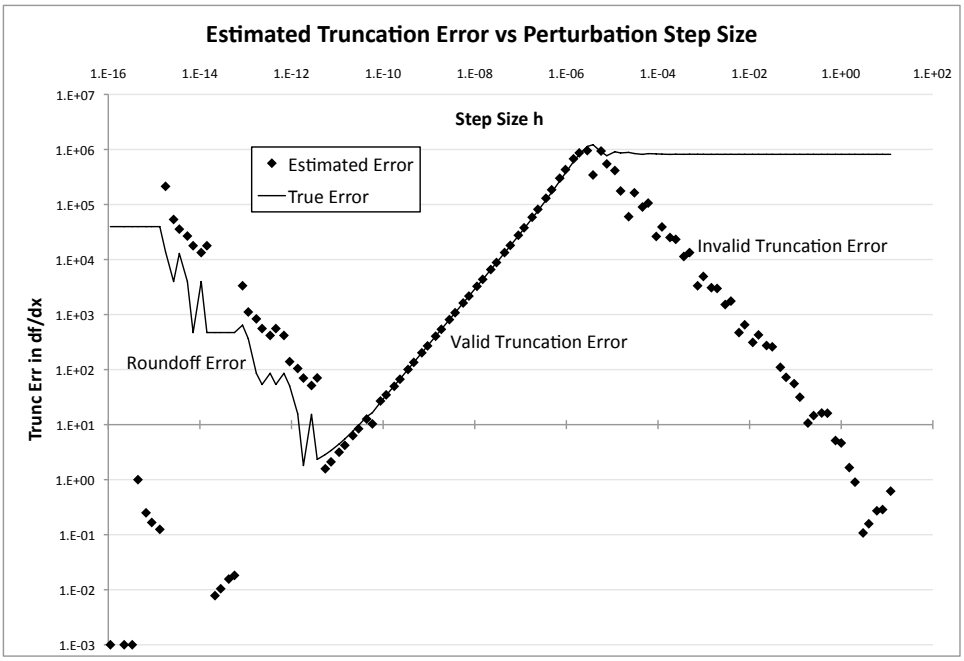


Figure 2.5: Comparison of estimated and true truncation errors for $\sin(x^2+10^6x)$, $x = \pi/4$.

$\sin(x^2 + 10^6x)$ – then this optimal step size can be computed via conventional approximations. However, with no prior knowledge of the function's qualities, it would be difficult to estimate that the optimal step size is a full 3 orders of magnitude away from the common wisdom choice. Because the implementation of the function of interest is assumed to be unknown, it is necessary to study not only the truncation error, but also to form estimates for the roundoff errors in the FDD equations.

## 2.7 Roundoff Error Estimation

In the simplest sense, roundoff error is an error in the computation of a number caused by the fact that the number is represented using finite numerals. Although the term 'roundoff error' has so far been referred to in a general sense, there are actually multiple sources of roundoff errors [15]. Because these errors are often tiny when considered individually, their analysis is sometimes neglected. Doing so can have disastrous results; an extreme example is the Patriot missile failure of 1992 [49]. A software oversight in converting numbers between finite-precision representations caused a Patriot missile system to lag by 0.3433 seconds. In this short time, an enemy Scud missile being tracked was able to travel an extra 1/2 kilometer, hit a barracks, and kill 28 soldiers.

While the consequences of this dissertation may not be so dire, a proper understanding of roundoff error sources nevertheless provides useful tools for step-size analysis. Two types of roundoff error are considered here: cancellation error and condition error.

### 2.7.1 Cancellation Error

Given two numbers $a$ and $b$ represented in fixed precision, cancellation error occurs when a significant number of the leading digits of $a$ and $b$ are equal and the two numbers are subtracted. Consider two numbers $a$ and $b$ with values

$$a = 0.3142049 \quad b = 0.3141550 \quad (a-b)_{\text{true}} = 0.0000499$$

If a 5-digit fixed precision representation (with standard rounding) is used to approximate $a$ and $b$, then the subtraction of the numbers becomes,

$$a = 0.31420 \quad b = 0.31416 \quad a - b = 0.00004$$

This subtraction operation has error in the least significant digit of the result, as compared to the true result. In general, fixed precision subtraction results in an error whose magnitude is at most that of the least significant digit in the larger of the two numbers [15]. An accurate upper bound to this error is,

$$|(a-b)_{\text{true}} - (a-b)| \leq \delta \max(|a|, |b|) \tag{2.35}$$

where $\delta$ is the precision of the representation. In the above example with 5-digit precision, $\delta = 10^{-5}$. For a standard double precision representation on a computer (defined by IEEE 754 [22, 48]), $\delta = 2^{-53}$. It is important to note that (2.35) only gives an upper bound on subtractive cancellation error; there is no way to know the exact error without additional information about the computation of $a$ and $b$.

**Example 2.1.** For FDD approximations, cancellation error enters via the subtraction between various function evaluations. Consider the forward-difference approximation (2.19) for $f(x) = \sin(x)$ at $x = \pi/4$, as shown in Figure 2.3. The approximation of the first derivative is

$$FD_1^{(1)}(x, h) = \frac{\sin(x+h) - \sin(x)}{h} \tag{2.36}$$

Because the evaluation of $\sin(x)$ is accurate to full precision for reasonable values of $x$, the subtraction in the approximation is subject only to cancellation error as

described in (2.35). Using this equation, the error can be approximated as,

$$|FD_{\text{true}} - FD| = \left| \frac{[\sin(x+h) - \sin(x)]_{\text{true}} - [\sin(x+h) - \sin(x)]}{h} \right| \tag{2.37}$$

$$\leq \frac{\delta \max(|\sin(x+h)|, |\sin(x)|)}{h} \tag{2.38}$$

where the sub and superscripts have been dropped from $FD$ for conciseness. Assuming that the step size $h$ is sufficiently small, the magnitudes of $\sin(x+h)$ and $\sin(x)$ will be approximately equal. The error in the FDD then becomes,

$$|FD_{\text{true}} - FD| \leq \frac{\delta|\sin(x)|}{h} \tag{2.39}$$

Since $\delta$ is a constant and $\sin(x)$ is independent of the step size, it is clear to see that as $h$ decreases, the cancellation error in the FDD approximation will increase. This is consistent with the roundoff error portion of Figure 2.3.

### 2.7.2 Condition Error

During cancellation error analysis in the preceding section, it was assumed that the values being subtracted were known to machine precision. In the example given, the values were computed using the sin() function, which is indeed accurate to machine precision in all modern implementations. However, in general a function may not have a perfectly precise output value (in the numerical sense). For example, a function may numerically integrate its input in order to produce its output. This numerical integration will likely have a specified global error tolerance, meaning that only a certain number of significant digits in its output will be accurate. In addition, the algorithm implemented within the function may consist

of many elementary operations, each of which may accumulate a small amount of error. An example is the solution of Kepler's equation [2]. Because this equation is transcendental, common solution methods are iterative and have specified convergence tolerances. It is shown in Chapter 4 that certain implementations of Kepler's equation can introduce significant error into the final result.

Errors in the output of a function, even if the input is exactly correct, are called condition errors. Because they reduce the number of accurate digits in a function's output, condition errors can result in significant loss of accuracy in a FDD computation of that function.

It is important to distinguish the terms *condition error* and *condition number*. The relative condition number $\hat{\kappa}$ of a single-valued function $f(x)$ is defined as [53],

$$\hat{\kappa}(f,x) = \left| x\frac{f'(x)}{f(x)} \right| \tag{2.40}$$

The condition number of a function predicts how small errors in a particular input value will affect the output value of the function. This definition implies that a condition number is associated with a problem itself, not necessarily with its implementation. While the condition number of a problem and condition error of its implementation are not unrelated, only the latter is of significance to this dissertation.

**Example 2.2.** The condition number for the function $\sin(x)$ at $x = 10^6$ is

$$\hat{\kappa} = \left| 10^6 \frac{cos(10^6)}{\sin(10^6)} \right| = 2.7 \times 10^6 \tag{2.41}$$

Although this condition number is very large, the value of $\sin(10^6)$ as computed by most modern implementations is accurate to machine precision, which implies a very small condition error for $\sin(x)$.

The step-size estimation methods of Gill et al. [12,13] and Barton [1] explicitly require an estimate of the condition error associated with the function being differentiated. Although there are methods to roughly approximate this condition error [19], the goal of this dissertation is to require as little prior information and estimation as possible.

Assuming that condition error affects all digits in the output of $f(x)$ below a certain threshold, an approximation of the upper bound of the error is given as,

$$|f(x)_{\text{true}} - f(x)| \leq \epsilon |f(x)| \tag{2.42}$$

where $\epsilon$ indicates the magnitude of the most significant digit affected by condition error. Although $\epsilon$ is equal to machine precision for elementary operations and many built-in functions (e.g. sin, cos, etc...), in general it is an unknown to be computed. Conversely, once $\epsilon$ is computed, it can be of great assistance in determining the amount of condition error introduced by a given implementation of $f(x)$. Because $\epsilon$ is a relative value indicating only which digits are erroneous, it allows for a meaningful comparison of condition errors between various implementations of the same problem.

**Example 2.3.** A function $f(x)$ computes its output by numerically integrating $x$ with a specified relative global error tolerance of $10^{-12}$. Because the accuracy of the output is only guaranteed for 12 digits, the condition error of $f(x)$ is $\epsilon = 10^{-12}$.

### 2.7.3 Total Roundoff Error

Although the exact value of roundoff error cannot be estimated for a general function, an estimate of the upper bound of roundoff error can be made. The roundoff error bounds in a finite-difference computation are formed using the expressions for cancellation and condition errors given in (2.35) and (2.42). For the forward-difference derivative approximation given in (2.19), using condensed notation,

$$FD_1^{(1)}(x,h) = FD = \frac{f(x+h) - f(x)}{h} = \frac{f_1 - f_0}{h} \tag{2.43}$$

where subscripts for $f$ are used to indicate the step size as a multiple of $h$. To develop an estimate of the maximum total roundoff error in the computation of this FDD, both cancellation and condition errors are handled independently.

The upper bound on cancellation error, caused by the subtraction of function values, is computed using (2.35) as

$$FD_{\text{true}} - FD = \frac{(f_1 - f_0)_{\text{true}} - (f_1 - f_0)}{h} \tag{2.44}$$

$$|FD_{\text{true}} - FD| \leq \frac{\delta |f_{1/0}|}{h} \tag{2.45}$$

$$|f_{1/0}| = \max(|f_1|, |f_0|) \tag{2.46}$$

where $FD_{\text{true}}$ indicates the true value of the finite difference derivative, in the absence of any roundoff errors. This is distinct from the true derivative itself, which is in general unknown.

The upper bound on condition error, caused by errors accumulated within

the function implementation, is computed using (2.42) as

$$FD_{\text{true}} - FD = \frac{(f_{1,\text{true}} - f_1) - (f_{0,\text{true}} - f_0)}{h} \tag{2.47}$$

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon_1|f_1| - \epsilon_0|f_0|}{h} \tag{2.48}$$

If $h$ is small enough such that the code paths taken by $f(x)$ and $f(x+h)$ are the same, then the magnitudes of the relative errors $\epsilon_0$ and $\epsilon_1$ introduced by both calls to $f$ will be approximately equal. In addition, for the worst case, the signs of the errors will be opposite, causing the errors to compound. The condition error then simplifies to,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_0|)}{h} \tag{2.49}$$

In contrast, if $h$ is large enough such that the implementation of $f$ uses different code paths (with different relative errors) to compute $f(x)$ and $f(x+h)$, then this simplification does not hold. It is assumed here that this situation will not occur when $h$ is near its optimal value, which is in general small relative to $x$.

In the general case, both cancellation and condition errors affect the upper bound of the error in the finite-difference approximation. The total error bound due to roundoff errors is the sum of both error bounds,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_0|) + \delta|f_{1/0}|}{h} \tag{2.50}$$

$$|f_{1/0}| = \max(|f_1|, |f_0|) \tag{2.51}$$

where it is assumed that $\epsilon$ is unknown.

For the central-difference derivative approximation given in (2.24), using condensed notation,

$$FD_2^{(1)}(x,h) = FD = \frac{f(x+h) - f(x-h)}{2h} = \frac{f_1 - f_{-1}}{2h} \tag{2.52}$$

Using a formulation similar to the forward-difference case above, the total error bound due to roundoff errors is,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_{-1}|) + \delta|f_{\pm 1}|}{2h} \tag{2.53}$$

$$|f_{\pm 1}| = \max(|f_1|, |f_{-1}|) \tag{2.54}$$

Both (2.50) and (2.53) follow the general form of the roundoff error bounding equation,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} \tag{2.55}$$

where $|F_\epsilon|$ and $|F_\delta|$, derived from the finite-difference expression $\Delta f_n^{(d)}$ in (2.25), are given for various FDD approximations in Appendix B. Note that any constant which traditionally appears in the denominator is now absorbed into the $|F_\epsilon|$ and $|F_\delta|$ expressions.

## 2.8 Total Error Estimation

The total error for a particular FDD approximation and function implementation can be bounded using the expressions for truncation error and roundoff error. The general FDD equation given by (2.28) is first rewritten using condensed notation,

$$f_{\text{true}}^{(d)} = FD_{\text{true}} + C_{\text{true}} h^n \tag{2.56}$$

where the true finite-difference derivative in the absence of roundoff errors, $FD_{\text{true}}$, is unknown. Subtracting the computed FDD, which contains roundoff errors, gives the total error in the finite-difference derivative,

$$f_{\text{true}}^{(d)} - FD = FD_{\text{true}} - FD + C_{\text{true}} h^n \tag{2.57}$$

Taking the magnitude of the total error and applying the triangle inequality gives an upper bound on the total error,

$$|f_{\text{true}}^{(d)} - FD| = |FD_{\text{true}} - FD + C_{\text{true}} h^n| \tag{2.58}$$

$$\leq |FD_{\text{true}} - FD| + |C_{\text{true}} h^n| \tag{2.59}$$

If the step size $h$ is small enough such that (2.30) applies, then the true value of the truncation error coefficient $C_{true}$ is well approximated by $C_n$ as given in (2.33). In addition, the roundoff error can be substituted from (2.55), giving the equation for the total error bound,

$$|f_{\text{true}}^{(d)} - FD| \leq \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n| h^n \tag{2.60}$$

where the explicit expression for $|FD_{\text{true}} - FD|$ is given by (2.50), (2.53), or the appropriate roundoff error bounding equation from Appendix B. For the $2^{nd}$-order central-difference approximation of the first derivative,

$$|f'_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_{-1}|) + \delta|f_{\pm 1}|}{2h} + |C_2| h^2 \tag{2.61}$$

A method to reliably compute the condition error $\epsilon$ is presented later in this dissertation, but $\epsilon$ can also be treated as a parameter to observe its effects on the total error bound.

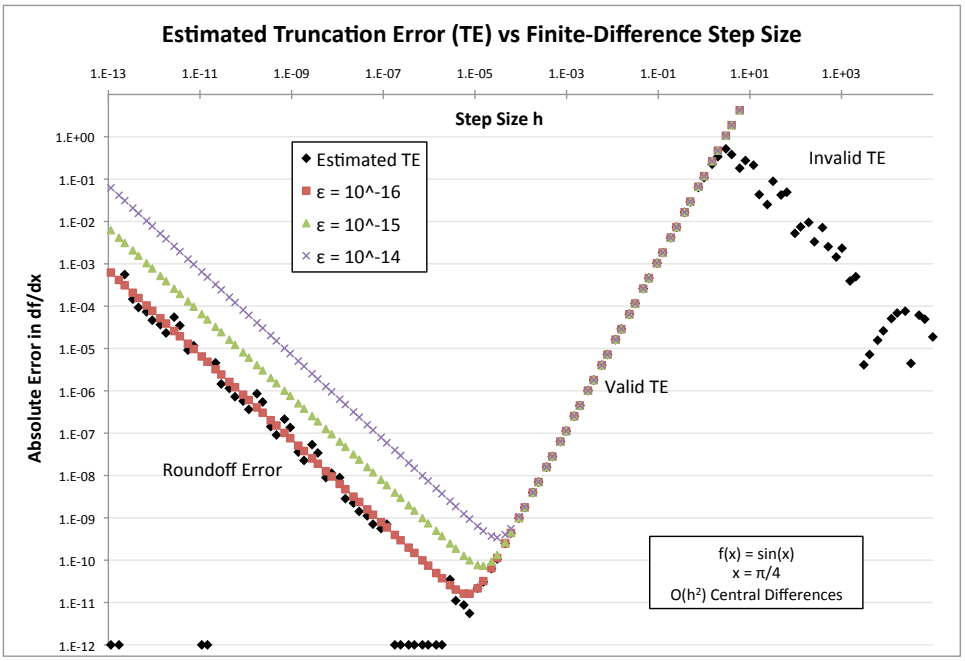


Figure 2.6: Effect of condition error $\epsilon$ on the total error bound.

**Example 2.4.** For the function $f(x) = \sin(x)$ with $x = \pi/4$ and using the central-difference approximation, the effect of condition error $\epsilon$ is seen in Figure 2.6. The total error bound from (2.61) is plotted for various values of $\epsilon$, and the estimated truncation error from (2.34) is also plotted for comparison. $C_2$ is computed as $\frac{-1}{3!}f^{(3)}(x)$, and $\delta = 2^{-53}$. It can be seen that in the roundoff error range of step sizes, the $\epsilon$ value for which the total error bound most closely approximates the estimated truncation error is $\epsilon = 10^{-16}$. This agrees with the fact that the sin() function is accurate to machine precision. Furthermore, the $\epsilon = 10^{-16}$ curve minimizes at almost exactly the same step size value as the estimated truncation error curve.

This example shows that if $\epsilon$ is known, then the optimal step size can be computed as the minimizing $h$ of the total error bound curve. This result is not new; it forms the basis of existing step-size estimation algorithms [1, 12, 13, 51]. Of greater interest in this dissertation is the converse fact that if an optimal step size can be found using the truncation error estimate curve, then the appropriate form of (2.60) can be used to compute the function's condition error with no prior information about the function. To do so, the total error function $E(x, h)$ is first defined as the right-hand side of (2.60),

$$E(x,h) = \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n|h^n \tag{2.62}$$

Note that the point of differentiation $x$ does appear explicitly in this equation; it is used to compute $|F_\epsilon|$ and $|F_\delta|$.

To properly use (2.62), both $|F_\epsilon|$ and $|F_\delta|$ must be evaluated at the optimal step size $h_{\text{opt}}$, and $C_n$ must closely approximate $C_{\text{true}}$. The latter condition is satisfied if the step size is sufficiently small and (2.30) applies. The former condition can be satisfied either by specifying $\epsilon$ and estimating $|F_\epsilon|$ and $|F_\delta|$, or by specifying the optimal step size. In either case, (2.62) is solved not directly, but by noting that the optimal step size occurs at its minimum,

$$\frac{\partial E}{\partial h} = -d\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^{d+1}} + n|C_n|h^{n-1} \tag{2.63}$$

where, to be clear, $d$ is the derivative order being computed and $n$ is the truncation

error order of the FDD method used. Solving this derivative at the optimal $h_{\text{opt}}$,

$$\left.\frac{\partial E}{\partial h}\right|_{h_{\text{opt}}} = 0 \tag{2.64}$$

$$-d\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h_{\text{opt}}^{d+1}} + n|C_n|h_{\text{opt}}^{n-1} = 0 \tag{2.65}$$

If the condition error $\epsilon$ is known, then approximations for $|F_\epsilon|$ and $|F_\delta|$ can be used to solve for the optimal step size,

$$h_{\text{opt}} = \left[\frac{d}{n}\frac{1}{|C_n|}(\epsilon|F_\epsilon| + \delta|F_\delta|)\right]^{1/(n+d)} \tag{2.66}$$

This equation simplifies to the approximations of Gill et al. [12,13] in the case of the first derivative ($d = 1$) using the forward-difference approximation ($n = 1$). Particular forms of this equation for various derivatives and orders are given in Appendix B.

If the condition error $\epsilon$ is not known, but the optimal step size $h_{\text{opt}}$ can be determined, then the condition error can be computed as

$$\epsilon = \frac{1}{|F_\epsilon|}\left(\frac{n}{d}|C_n|h_{\text{opt}}^{n+d} - \delta|F_\delta|\right) \tag{2.67}$$

## 2.9 Accuracy of the Estimated Truncation Error

It has so far been assumed that the truncation error estimate $TE_n(x, h)$ given in (2.33) and (2.34) accurately models both truncation and roundoff errors. Given a step size $h_1$ and a comparison step size $h_2 < h_1$, the truncation error estimate for a finite-difference approximation of the $d^{th}$ derivative using an $n^{th}$-order Taylor polynomial is

$$TE_n(x, h_1) = \frac{FD_n^{(d)}(x, h_2) - FD_n^{(d)}(x, h_1)}{h_1^n - h_2^n}h_1^n \tag{2.68}$$

For large step sizes, it was shown in Figure 2.4 that this is not an accurate estimate of the true truncation error. For step sizes which are sufficiently small, but not so small as to incur significant roundoff error, the estimate is mathematically designed (in Section 2.6) to closely model truncation error. It is now proven that (2.68) does indeed accurately model roundoff error when the step size is very small.

In practice, the ratio of step sizes $h_2/h_1$ is often kept constant in order to reduce coding complexity. This step-size reduction ratio $t$, defined as $t = h_2/h_1$ with $0 < t < 1$, is used to reformulate the truncation error estimate.

$$TE_n(x, h_1) = \frac{FD_n^{(d)}(x, h_2) - FD_n^{(d)}(x, h_1)}{1 - t^n} \tag{2.69}$$

This result only applies if roundoff errors do not affect the computation of $FD_n^{(d)}$. If roundoff errors are considered, then the error in the truncation error equation is,

$$TE - TE_{\text{true}} = \left(\frac{FD_2 - FD_1}{1 - t^n}\right) - \left(\frac{FD_2 - FD_1}{1 - t^n}\right)_{\text{true}} \tag{2.70}$$

$$= \frac{(FD_2 - FD_{2,\text{true}}) - (FD_1 - FD_{1,\text{true}})}{1 - t^n} \tag{2.71}$$

$$|TE - TE_{\text{true}}| \leq \frac{|FD_{2,\text{true}} - FD_2| + |FD_{1,\text{true}} - FD_1|}{1 - t^n} \tag{2.72}$$

where condensed notation has been used, and the triangle inequality was employed in the last equation. For very small step sizes (i.e. $h < h_{\text{opt}}$), roundoff error dominates the FDD computation, and (2.55) can be used to express the right-

hand side.

$$|TE - TE_{\text{true}}| \leq \frac{1}{1-t^n}\left[\frac{\epsilon_2|F_{\epsilon_2}| + \delta_2|F_{\delta_2}|}{h_2^d} + \frac{\epsilon_1|F_{\epsilon_1}| + \delta_1|F_{\delta_1}|}{h_1^d}\right] \tag{2.73}$$

$$\leq \frac{1}{1-t^n}\left[\frac{(1/t)^d(\epsilon_2|F_{\epsilon_2}| + \delta_2|F_{\delta_2}|) + (\epsilon_1|F_{\epsilon_1}| + \delta_1|F_{\delta_1}|)}{h_1^d}\right] \tag{2.74}$$

As stated at the end of Section 2.8, the various $F_\epsilon$ and $F_\delta$ values are all evaluated at the optimal step size, so $F_{\epsilon_1} = F_{\epsilon_2}$ and $F_{\delta_1} = F_{\delta_2}$. Furthermore, $\delta$ itself only depends on the machine precision, so $\delta_1 = \delta_2$. Finally, the condition error of the function $f(x)$ depends on the base value of $x$, which is independent of the step size, so $\epsilon_1 = \epsilon_2$. Making these substitutions and using generic notation,

$$|TE - TE_{\text{true}}| \leq \frac{1 + (1/t)^d}{1-t^n}\left(\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d}\right) \tag{2.75}$$

The true value of the truncation error becomes negligible as the step size is reduced into the roundoff error range, so this equation is equivalent to,

$$|TE_n(x,h)| \leq \frac{1 + (1/t)^d}{1-t^n}\left(\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d}\right) \tag{2.76}$$

This equation gives an upper bound on the roundoff error when the estimated truncation error (2.68) is used with neighboring small step sizes $h_1 = h$ and $h_2 = th$. In comparison, the total error in a FDD approximation, given in (2.62), is

$$E(x,h) = \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n|h^n \tag{2.77}$$

The first term in this equation, which dominates when the step size is small, gives the true roundoff error bound. It is easily shown that the constant term in (2.76) is always greater than unity since $0 < t < 1$, and $d$ and $n$ are both $\geq 1$

(see Section 2.4). This means that the truncation error given by (2.68) slightly overestimates the true roundoff error by a nearly constant amount. Because the estimated truncation error follows the same trend as the true error with respect to the step size and differs only by a constant, (2.68) is hereby proven to accurately model roundoff error when the step size is very small.

To obtain the most accuracy from the truncation error (2.68), it should be corrected for small step sizes by dividing out the constant from (2.76). The true error function is then well approximated by the piecewise equation,

$$E(x,h) \approx \begin{cases} TE_n(x,h_1) & h \geq h_{\text{opt}} \\ \frac{1-t^n}{1+(1/t)^d} TE_n(x,h_1) & h < h_{\text{opt}} \end{cases} \tag{2.78}$$

There are of course other piecewise equations which would result in smoother fits near $h_{\text{opt}}$, such as a linear transition from the uncorrected to the corrected equations. However, the actual behavior of the truncation error estimate at the optimal step size is not of interest; rather, the behavior away from $h_{\text{opt}}$ is more indicative of the optimal step size. This is proven in the analysis of Chapter 3.

The result of applying (2.78) to the truncation error estimate for $\sin(x)$ at $x = \pi/4$ using the forward-difference approximation (originally in Figure 2.4) is shown in Figure 2.7. It can be seen that the corrected truncation error estimate more closely approximates the true roundoff error.

Because the estimated truncation error (2.68) overestimates the true round-off error for very small step sizes, it can be concluded that the step size which minimizes (2.68) is slightly greater than the optimal step size given in (2.66) which

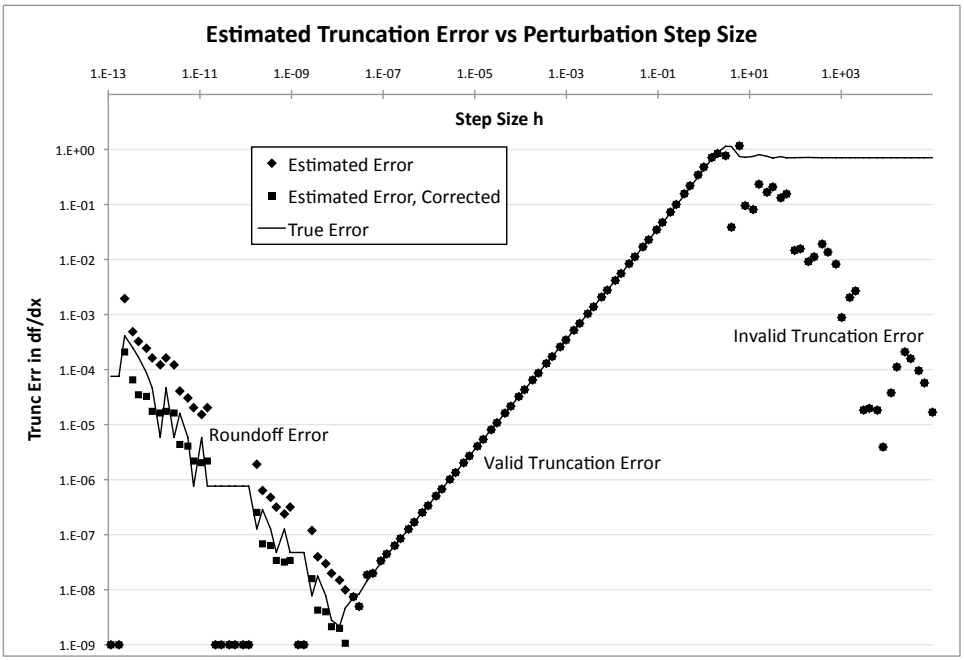


Figure 2.7: Result of correcting the truncation error estimate for $\sin(x)$, $x = \pi/4$.

minimizes the true error (2.60). This effect is shown in Figure 2.8, which analyzes the function $f(x) = sin(x^2 + 10^6 x)$, first seen in Figure 2.5. As expected, the corrected truncation error estimate points are much better fits to the true error. To visualize the optimal step size, best-fit dashed lines are shown for both uncorrected and corrected truncation error estimates. It is clear that the optimal step size resulting from the uncorrected truncation error is greater than from the corrected truncation error.

The relationship between the uncorrected and corrected optimal step sizes is derived from the fact that the truncation error estimate and the true error differ only by a proportionality constant for very small step sizes. The general form of

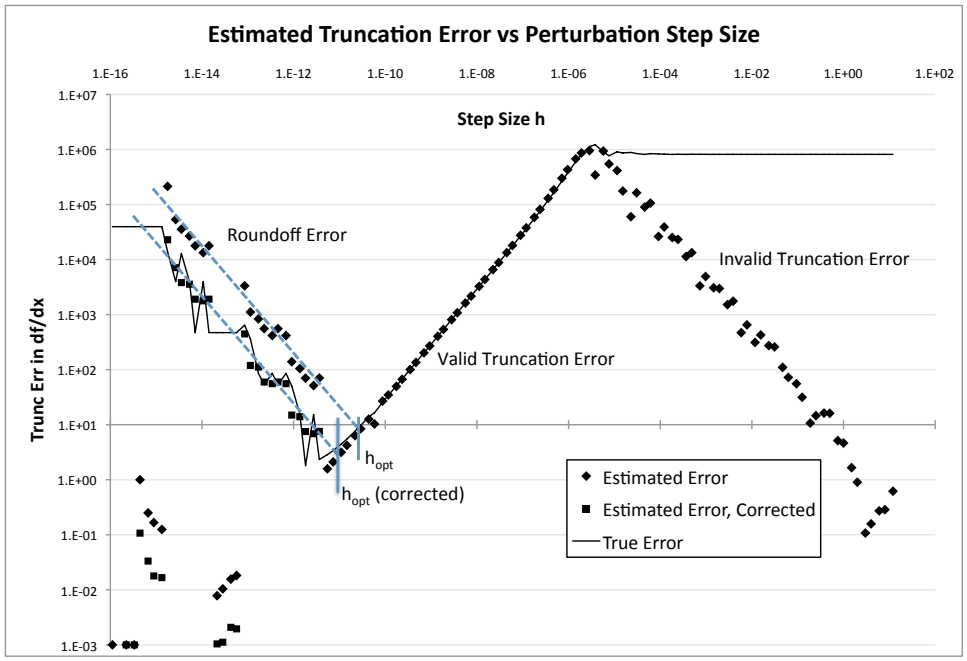


Figure 2.8: Optimal step size using a corrected truncation error estimate for $\sin(x^2 + 10^6 x)$, $x = \pi/4$.

the true error is,

$$E_{\text{true}} = \frac{e}{h^d} + Ch^n \tag{2.79}$$

while the general form of the estimated truncation error is,

$$E_{\text{TE}} = \frac{t^* e}{h^d} + Ch^n \tag{2.80}$$

where $e = \epsilon|F_\epsilon| + \delta|F_\delta|$, $C = |C_n|$, and $t^* = (1 + (1/t)^d)/(1 - t^n)$. Differentiating

both of these and solving for the optimal step size,

$$E'_{\text{true}} = -d\frac{e}{h^{d+1}} + nCh^{n-1} \qquad E'_{\text{TE}} = -d\frac{t^*e}{h^{d+1}} + nCh^{n-1} \tag{2.81}$$

$$h_{\text{opt,true}} = \left(\frac{d}{n}\frac{e}{C}\right)^{1/(n+d)} \qquad h_{\text{opt,TE}} = \left(\frac{d}{n}\frac{e}{C}t^*\right)^{1/(n+d)} \tag{2.82}$$

$$h_{\text{opt,true}} = \left(\frac{1}{t^*}\right)^{1/(n+d)} h_{\text{opt,TE}} \tag{2.83}$$

If the step-size reduction ratio is in the range $0 < t < 1$, then $t^* > 1$ and it is proven that the true optimal step size is smaller than the uncorrected optimal step size. This relationship is useful when the truncation error estimate is used to determine an optimal step size (as explained in the next chapter), which can then be easily corrected using (2.83).

It is interesting to analyze the relationship between $t$ and $t^*$, in particular with regards to making the uncorrected optimal step size as close to the true one as possible.

$$t^{*(d)}_{n}(t) = \frac{1 + (1/t)^d}{1 - t^n} \tag{2.84}$$

The range of $t^*$ with respect to $t$ is given in Figure 2.9 for various values of the FDD order $n$. Clearly there is a value of $t$ for which $t^*$ is minimized; i.e. the uncorrected step size $h_{\text{opt,TE}}$ will be as close to $h_{\text{opt,true}}$ as possible.

The exact value of $t$ which minimizes $t^*$ is computed as,

$$\frac{dt^*}{dt} = \frac{dt^{-d}(t^n - 1) + n(t^{-d} + 1)t^n}{t(t^n - 1)^2} = 0 \tag{2.85}$$

$$nt^{n+d} + (n + d)t^n - d = 0 \tag{2.86}$$

For a simple forward-differences approximation to the first derivative ($n = d = 1$), (2.86) is quadratic and solves to $t = -1 + \sqrt{2} \approx 0.41$. This is the only case in

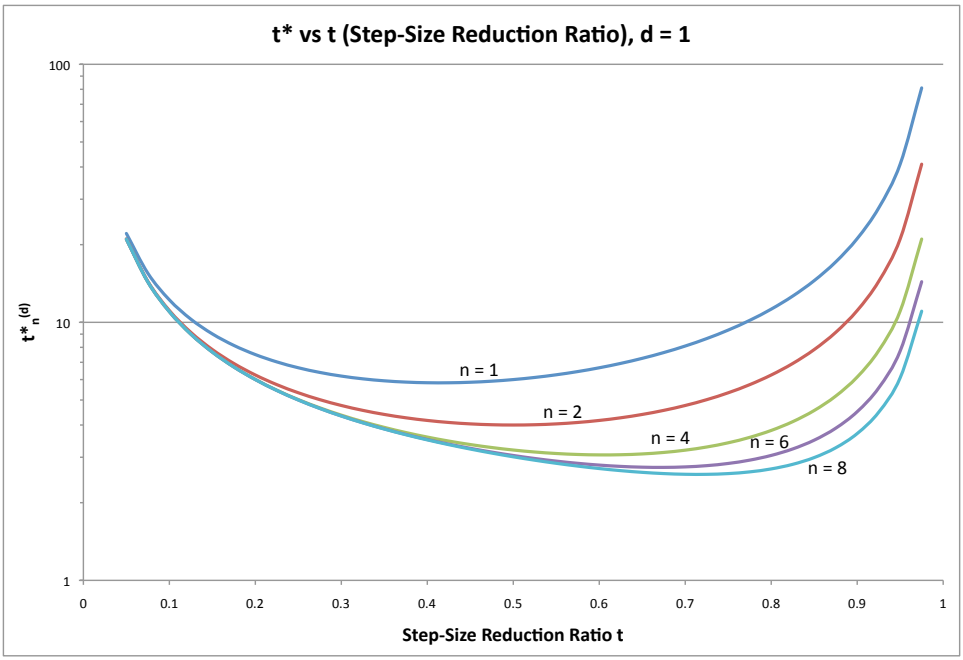


Figure 2.9: Effect of changing FDD order $n$ on $t^{*(d)}_{n}$.

which (2.86) is easily solved analytically; all other combinations of $d$ and $n$ require a numerical root-finding method.

The optimal step-size reduction ratio $t$ is computed for the derivative orders $d \in \{1, 2, 3, 4\}$ and common FDD orders $n \in \{1, 2, 4, 6, 8\}$. This $t$ is then used to evaluate the ratio $h_{\text{opt,TE}}/h_{\text{opt,true}}$ (which equals $t^{*1/n+d}$ from (2.86)), and the results for each $(d, n)$ combination are given in Figure 2.10. This graph shows that if the appropriate optimal step-size reduction ratio $t$ is chosen for a given $(d, n)$ pair, then the uncorrected optimal step size can in fact be quite close to the true optimal step size.

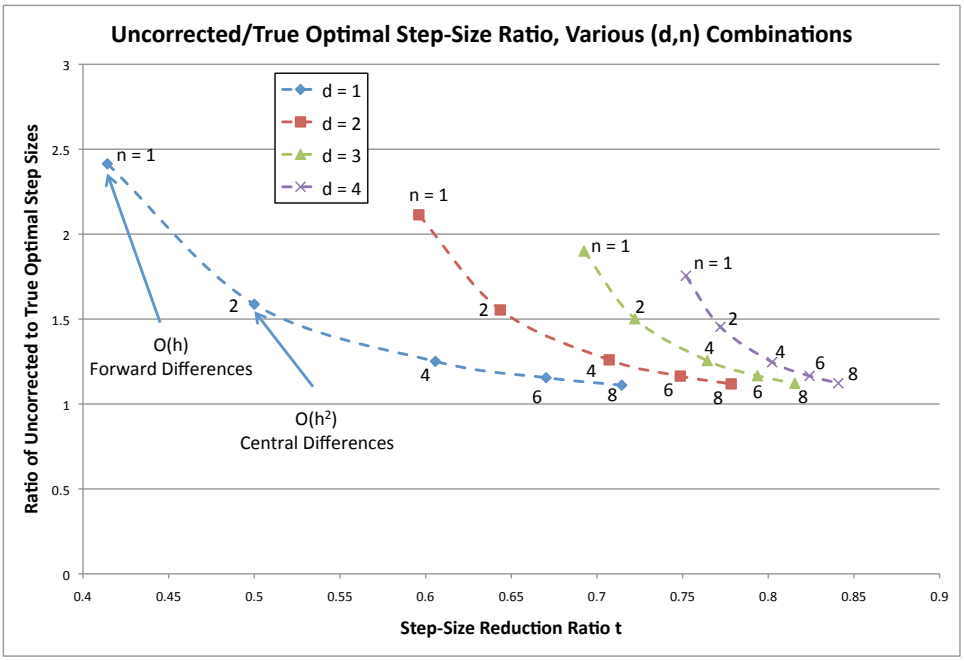


Figure 2.10: Effect of optimal step-size reduction ratio $t$ on the correction factor in 2.86.

## 2.10 Chapter Conclusions

The basics of Richardson Extrapolation, Taylor Series, the Lagrange remainder, and finite-difference derivative (FDD) methods were discussed at the beginning of this chapter. In particular, it was shown in Section 2.5 that if the step size used to compute a given FDD approximation is too small, then the error of the method increases in inverse proportion to the step size. This problem is called the 'Step-Size Dilemma', and is caused by roundoff errors interfering with the computations within the FDD approximation.

The two main sources of roundoff error affecting a FDD computation are cancellation and condition errors. By forming expressions to approximate the

upper bounds of these errors, equation (2.60) was developed which accurately approximates both roundoff and truncation errors. Because these errors occur in inverse proportions, it was shown that the total error bound can be differentiated, and the resulting zero gives the optimal step size $h_{\text{opt}}$ that minimizes total error in a FDD approximation. This optimal step size, given in (2.66), is a generalization of the ones given by other noted authors [1, 6, 12, 13, 19]. As noted by these authors, its use requires prior knowledge of the condition error in the function being differentiated.

Finally, a method to estimate the truncation error of a FDD approximation was developed by using a variation of Richardson Extrapolation. This truncation error estimate, given in (2.33) and (2.34), was shown in Section 2.9 to also correctly model roundoff error (with a constant correction factor) when the step size is below its optimal value.

Overall, mathematical tools have been developed which will be employed in the next chapter to search for the optimal step size, and to analyze the function of interest $f(x)$ once the optimal step size has been found.

# Chapter 3

# Developing the Step-Size Optimization Algorithm

## 3.1 Chapter Summary

This chapter focuses on developing a robust algorithm that determines the step size which minimizes both roundoff and truncation errors in the computation of a finite-difference derivative (FDD). The step-size analysis tools developed in Chapter 2 are used for this purpose, and are analyzed from a numerical perspective. A logarithmic analysis of the total error bound (2.60) sheds light on a unique method of finding the optimal step size. This method is compared and contrasted to the iterative search procedure of Stepleman and Winarsky [51], which simply searches a descending error series for the best step size.

The validity of a given step size is also discussed. This gives a measure of whether a given optimal step size is still close to optimal if the point of differentiation $x$ is changed. In particular, a bound on $x$ is developed for which the optimal step size is deemed valid.

Because roundoff error is inherently unpredictable (for the purposes of this dissertation), the term 'optimal step size' must be used with caution. Since $h_{\text{opt}}$ is computed using analytical approximations developed in Chapter 2, the effects

of varying it slightly are also explored.

## 3.2 Algorithm Goals

The most common step-size approximation algorithms [1, 12, 13] require knowledge of the condition error within a function. While the iterative algorithm of Stepleman and Winarsky [51] does not require this knowledge, it can get 'stuck' in false minima caused by roundoff error near the true optimal step size. This algorithm searches for the optimal step size by iteratively reducing the step size, and stopping when the change in the the resulting FDD values is no longer decreasing. In other words, the basic search condition used is

$$|FD(h_i) - FD(h_{i+1})| \leq |FD(h_i) - FD(h_{i-1})| \tag{3.1}$$

where $h_i$ is the current step size being tested, and $h_{i+1}$ and $h_{i-1}$ are the next (smaller) and previous (larger) step sizes, respectively. It was recognized that starting this search with too large of a step size would result in an immediate failure, so a heuristic method was presented to iteratively compute the initial step size. While this method suffices for most functions, it can certainly fail for out of the ordinary functions with higher condition errors.

The step size optimization algorithm developed in this chapter is designed to work with any function $f(x)$ that is analytic in the neighborhood of the point of differentiation $x$. No assumptions are made as to the condition of the function or the value of the optimal step size. As seen in Section 2.6, a function can always be conceived of which violates the traditional step-size estimation theories.

Furthermore, the algorithm developed here is iterative in nature. Starting with a sufficiently large step size, successively smaller step sizes are analyzed until the optimal one is found. A unique requirement of the algorithm is that it should not be averse to overly large initial step sizes. Given such a large step size, the algorithm should be able to overcome the initial erratic behavior of the resulting finite-difference derivative and continue to find the true optimal step size.

Because the analysis of each step size requires multiple function evaluations, the total number of function evaluations can get expensive. In the interest of a thorough analysis, the algorithm is initially developed without regard to a high number of function evaluations. Optimizing the algorithm to reduce function evaluation cost is considered in Chapter 5, after the algorithm is fully developed and examples are shown.

## 3.3 Benefit of Power-of-2 Step Sizes

From a mathematical perspective, the value of a step size is independent of how that step size is represented. Because people are trained to think in base 10, it is natural to use step sizes which are associated with powers of 10: $1e{-}6$, $5e{-}8$, etc... Step-size analysis theories such as the ones developed in Chapter 2 suggest that the total FDD error associated with a step size depends only on truncation error (from the Taylor Series), cancellation error (from the subtraction operation), and condition error (accumulated within the function itself). There is, however, one more numerical source of error: representation error.

Representation error arises from the fact that not all numbers can be ex-

actly expressed in binary using a fixed number of digits. In particular, *no* negative integer power of 10 has an exact binary representation. Although the error between a number and its binary representation is smaller than machine precision, it can have a nontrivial effect on FDD approximations, as evidenced by the following example.

**Example 3.1.** The derivative of $f(x) = x^2$ at $x = 1$ is computed using a second-order central-difference approximation, and the results for a range of step sizes are tabulated in Table 3.1. Since the function is quadratic and the FDD method is $2^{nd}$-order, there is no truncation error. In addition, both cancellation and condition errors are smaller than machine precision for a simple $x^2$ multiplication near unity. Finally, this code computes the 'true' step size to account for roundoff errors in computing the perturbed $x$ values, as discussed in Numerical Recipes [39]. As a result of these considerations, the derivatives computed by this code should equal the analytical derivative ($df/dx = 2$) for all tested step sizes. It can be seen in Table 3.1 that this is not the case; several derivatives have nontrivial errors.

Since all other sources of error have been eliminated, only representation error could cause the incorrect derivatives in Table 3.1. Even though small errors in the actual step size have already been accounted for in the code, the resulting perturbed test points are not always centered at $x$. As a consequence, there is a possibility that the derivative is computed not at the desired $x$, but at a slightly shifted point.

This form of representation error is easily avoided by using a step size that is a power of 2, for example $h = 2^{-p}$. Because such numbers have exact binary

Table 3.1: Representation error in the central-difference derivative.

| Step Size | $d(x^2)/dx,\ x = 1$ |
|---|---|
| $10^{-1}$ | 2.000000000000000 |
| $10^{-2}$ | 2.000000000000000 |
| $10^{-3}$ | 1.999999999999944 |
| $10^{-4}$ | 1.999999999999445 |
| $10^{-5}$ | 2.000000000000000 |
| $10^{-6}$ | 2.000000000055511 |
| $10^{-7}$ | 2.000000000000000 |
| $10^{-8}$ | 1.999999994448885 |
| $10^{-9}$ | 2.000000000000000 |
| ⋮ | ⋮ |
| $10^{-15}$ | 2.000000000000000 |

representations, both $x + h$ and $x - h$ will be correct to full precision, and will in fact be centered at the desired $x$. When example 3.1 is reproduced with power-of-2 step sizes, the resulting derivatives are all 2.0 (to machine precision).

Although the effects of representation error may not always be as drastic as in example 3.1, they are so easily avoidable that there is no reason for not using power-of-2 step sizes. Even if the step size is computed relative to $x$,

```
h = eps * (1.0 + abs(x));
```

the equivalent power-of-2 step size would be computed as,

```
h = eps * (1.0 + abs(x));
p = ln(h) / ln(2.0);
h = 2.0^round(p);
```

## 3.4 Slope of the Total Error

The total error function (2.62) can be analyzed from a logarithmic standpoint to reveal linear trends as the step size moves away from its optimal value.

$$\ln E(x,h) = \ln\left(\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n|h^n\right) \tag{3.2}$$

$$\ln E(x,h) = \begin{cases} \ln\left(\epsilon|F_\epsilon| + \delta|F_\delta|\right) - d\ln h & h \ll h_{\text{opt}} \\ \\ \ln|C_n| + n\ln h & h \gg h_{\text{opt}} \end{cases} \tag{3.3}$$

This reformulation indicates that the slope of the total error function, on a log-log scale, approaches the negative of the differentiation order $d$ as the step size gets small, and approaches the FDD order $n$ as it gets large. There are two caveats to this rule. First, because roundoff errors are unpredictable, the limit for small step sizes acts as a best-fit line; the true errors exhibit small fluctuations about this line. Second, the limit for large step sizes only applies up until the point where the $n^{th}$-order truncation error estimate is no longer valid. As proven in Section 2.9, the truncation error estimate from (2.68) accurately models the true error when used with the correction factor in (2.78). Therefore, it is subject to the same linear trends regarding small and large step sizes.

**Example 3.2.** The 1-$\sigma$ Ricker wavelet commonly used in statistical analysis and also tested by Pemba [37],

$$f(x) = \frac{2}{\sqrt{3}\pi^{1/4}}\left(1 - x^2\right)e^{-x^2/2} \tag{3.4}$$

is differentiated at $x = e$ using the forward-difference approximation. Figure 3.1 both confirms and alleviates concerns with the first caveat associated with the

slope of the estimated truncation error for small step sizes. It is seen that the best-fit line for the corrected truncation error points (in the roundoff error region) does in fact have a slope very close to the predicted value of $-d = -1$. In addition, the best-fit line in the valid truncation error region is such an exact match to its constituent points and to the true truncation error that it is indistinguishable from both.

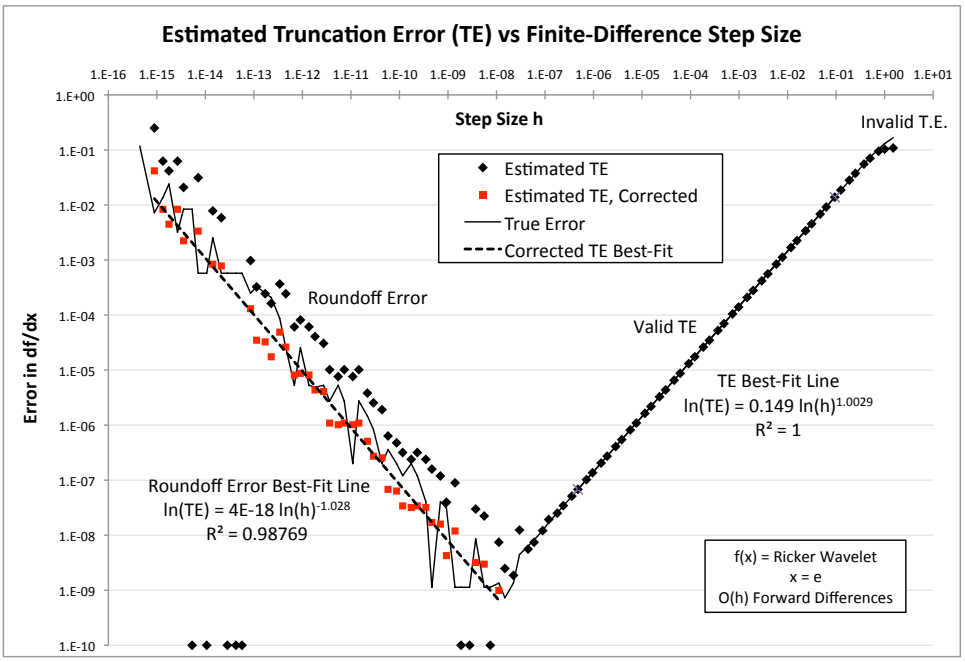


Figure 3.1: The slope of the estimated truncation error best-fit line approximates the true roundoff error.

It was shown in Section 2.6 that the estimated truncation error is not a valid approximation for step sizes that are too large. This is confirmed by performing error analysis on the truncation error estimate (2.34). Expanding the general FDD equation in (2.28) to include its error term and applying the Richardson

extrapolate,

$$f' = FD_1 + C_n h_1^n + O(h_1^{2n}) \tag{3.5}$$

$$f' = FD_2 + C_n h_2^n + O(h_2^{2n}) \tag{3.6}$$

$$C_n = \frac{FD_2 - FD_1}{h_1^n - h_2^n} + O(h_1^n) \tag{3.7}$$

$$TE_n(x, h_1) = C_n h_1^n + C_{2n} h_1^{2n} \tag{3.8}$$

where $C_{2n}$ depends on higher-order derivatives. On a log-log scale, the truncation error at first glance appears to have a predictable slope,

$$\lim_{h_1 \to \infty} \ln TE_n = \ln C_{2n} + 2n \ln h_1 \tag{3.9}$$

where the slope is $2n$. However, because $C_{2n}$ involves higher derivatives using the Lagrange Remainder form, it is evaluated over an ever-increasing interval of $[x - a_2 h_1, x + a_3 h_1]$ (from the general FDD approximation form of (2.29)). This causes $C_{2n}$ to change unpredictably as $h_1$ continues to increase. The overall consequence of this is that the estimated truncation error does not have a predictable slope (on a logarithmic scale) for step sizes far greater than the optimal. When used to estimate the true error, the estimated truncation error $TE(x, h)$ is therefore proven to not accurately model any of the true error's trends for such huge step sizes.

## 3.5 Algorithm Development for Scalar Functions

The step-size theory laid out in Chapter 2, combined with the analysis performed so far in this chapter, allows the development of a simple iterative step-size search algorithm.

### 3.5.1 A Simple Step-Size Search Algorithm

A 2-body orbit about the Earth, with orbital parameters given in Table 3.2, is propagated for a quarter of its period from a fixed initial true anomaly of $\nu_0 = 0°$. Danby's method[1] [7] is used to solve Kepler's equation, and the final result is the true anomaly $\nu_f$ at the final time $t_f$. Figure 3.2 shows the estimated truncation error when the derivative $d\nu_f/dt_f$ is computed using the second-order central-difference method.

Table 3.2: Orbital parameters for propagated orbit.

| | |
|---|---|
| $\mu$ | $398600.4\,[km^3/s^2]$ |
| $a$ | $200000\,[km]$ |
| $e$ | 0.96453 |
| $i$ | 51.619° |
| $\omega$, $\Omega$, $\nu_0$ | 0° |
| $(t_0, t_f)$ | $(0, 222533.8)\,[s]$ |

A simple algorithm which satisfies all the conditions in Section 3.2 can be developed by visually analyzing Figure 3.2 as follows.

1. A large initial step size $h_0$ is chosen, preferably one that is a power of 2 as explained in Section 3.3. If $h_0$ is too large, then the estimated truncation errors will be invalid, and it has been proven that in such cases the estimated truncation error slope will *not*, in general, be $n$. However, it has been observed that in isolated cases, a very large step size may result in a

[1] A useful summary of Danby's method for solving Kepler's equation is given at http://www.cdeagle.com/ommatlab/toolbox.pdf.

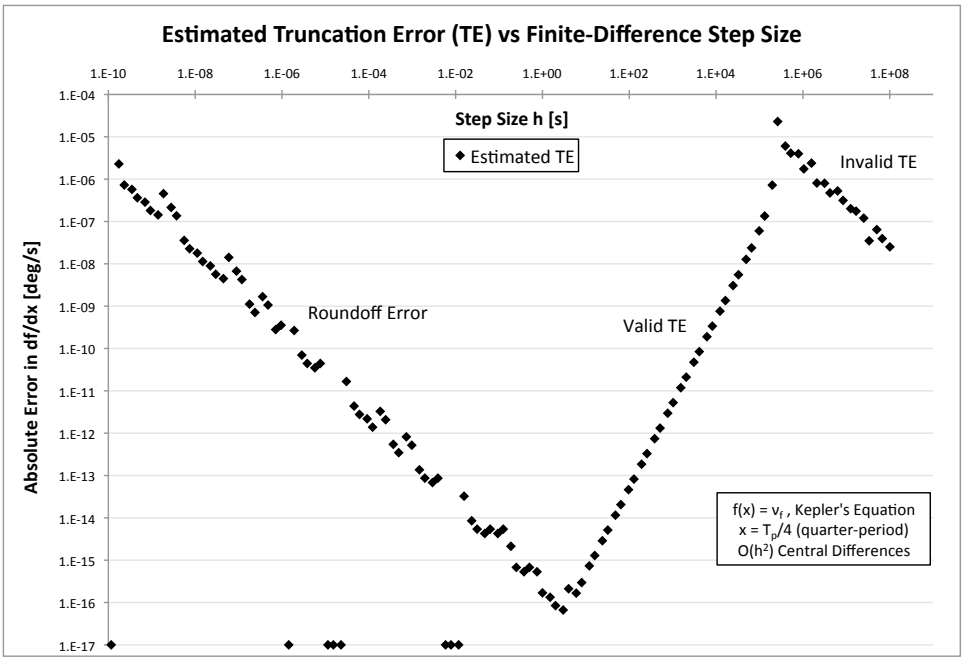


Figure 3.2: A truncation error plot used to develop the simple algorithm.

truncation error slope nearly equal to $n$ by coincidence. These cases occur sporadically; it may occur for a particular $h_i$ and $h_{i+1}$, but will only occur for a prolonged sequence of step sizes if the truncation error is actually valid.

2. A step-size reduction ratio $t$ is chosen, which relates two consecutive tested step sizes by $t = h_2/h_1$. It is known from Section 3.4 that the estimated truncation errors associated with these step sizes should have a slope equal to the FDD order (on a log-log scale), which in this case is $n = 2$. In addition, $t$ should be chosen as an inverse power of 2 so that $h_2$ will also be a power of 2.

3. The current step size $h_i$ and next step size $h_{i+1} = th_i$ are used with (2.68)

to compute a truncation error estimate. This is compared to the previous truncation error estimate, and the resulting slope is compared to the desired slope $n$. To accommodate the anomalous cases described in step 1, a counter variable is used to keep track of the number of consecutive step sizes with a near-correct truncation error slope. If the current slope is correct, then the counter is incremented and control goes to step 4. Otherwise, control goes to step 6.

4. If the counter has passed a predetermined limit, then a sufficient number of consecutive test step sizes have a truncation error slope equal to $n$. It is therefore assumed that the 'valid truncation error' range of step sizes has been reached; a flag is set to indicate this and control goes to step 5. However, if the counter has not yet reached its limit, then control returns to step 3 without setting the 'valid truncation error' flag.

5. Because the current step size results in a valid truncation error estimate, the associated $C_n$ value from (2.33) is saved for later use. Control returns to step 3.

6. If the slope from step 3 is not correct and the 'valid truncation error' flag is not set, then the counter is reset to zero and control returns to step 3. However, if the 'valid truncation error' flag is set, then it is assumed that roundoff error has caused the truncation error slope to deviate and control goes to step 7.

7. When control reaches this step, it is assumed that the current step size $h_i$ is the optimal uncorrected step size. Although not absolutely necessary, it is prudent to apply the step-size correction from (2.83): $h_{\text{opt}} = (t^*)^{-1/(n+d)} h_i$. Using this optimal step size and the previously saved $C_n$ value, the condition error $\epsilon$ can be computed using (2.67). The algorithm exits with the optimal step size.

Steps 3-5 of this algorithm create a logic which skips over any initial truncation error inaccuracies, recognizes the region of valid truncation error, and terminates at the first sign that roundoff error has begun to dominate the FDD. The optimal step size is then used to compute the condition error of the function. For the propagation problem in Figure 3.2, the solution computed by this algorithm is given in Table 3.3. Note that the corrected step size is adjusted to the closest power of 2 which has already been tested, to reduce function evaluations.

Table 3.3: Solution for Kepler orbit propagation problem.

| | uncorrected | corrected |
|---|---|---|
| $h_{\text{opt}}$ [s] | 4.0 | 3.0 |
| $d\nu_f/dt_f$ [deg/s] | $6.94245608057e{-7}$ | $6.94245607964e{-7}$ |
| Relative Error | $1.402e{-10}$ | $6.98e{-12}$ |

In both corrected and uncorrected cases, the function condition error $\epsilon$ is computed to be smaller than machine precision ($2^{-53}$) and the total number of function calls is 103. The relative error is computed using the true derivative, which is $6.94245607959e{-7}$ [deg/s]. Figure 3.3 illustrates the results of this problem; the

corrected and uncorrected optimal step sizes are indicated. The true optimal step size occurs at the bottom of the best-fit line, which (visually) is almost coincident with the corrected optimal step size. A final point of interest is the accuracy of the roundoff error best-fit line; it fits the true and corrected truncation errors extremely well, and its slope is almost exactly $-1$ as expected from this problem. This observation helps to validate the theory that corrected truncation error does a very good job of approximating true error in the roundoff region of step sizes.

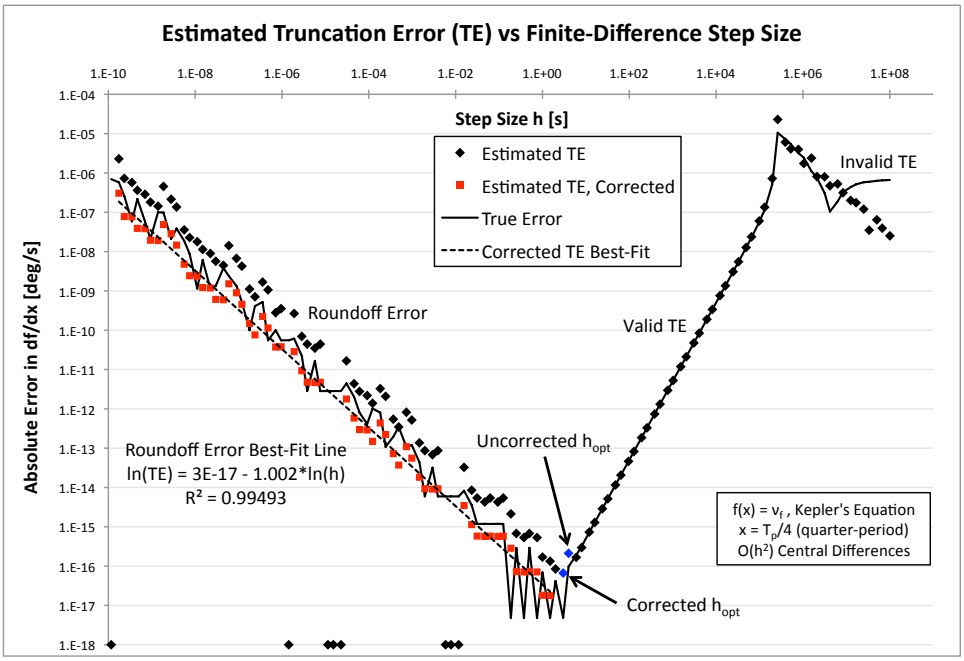


Figure 3.3: The simple algorithm applied to the Kepler orbit propagation problem.

### 3.5.2 Optimal Step Size Accuracy vs Precision

In step 7 of the simple step-size search algorithm, it is assumed that the optimal step size has been reached as soon as the truncation error slope deviates

from its expected value. Because deviations from the expected value will always exist, the algorithm must be specific as to how much deviation is considered excessive. If this allowable deviation is very small, then the algorithm may terminate several iterations early; i.e. the error would be smaller had it continued for a few more iterations. On the other hand, if the deviation tolerance is large, then the algorithm may iterate into step sizes where roundoff error dominates.

This uncertainty is exacerbated by the fact that, even if the chosen step size truly does result in a minimum error, it will only be a minimum for the given point of differentiation $x$. If the value of $x$ is changed by a slight amount $\Delta x$, it is useful to understand how the error in the computed FDD changes for a given step size $h$. There are some conclusions that can be reached by examining step-size optimization theory alone. For example if the test step size $h$ is slightly greater than the optimal step size $h_{\text{opt}}$, then roundoff errors have not yet begun to significantly effect the estimated error. Because of this, it is reasonable to expect that the estimated truncation error from the current $x$ will also bound the true error for a slightly different $x$. Conversely, if the test step size $h$ is slightly less than $h_{\text{opt}}$, then roundoff errors have a significant effect on the estimated error. In this case the true error for a slightly different $x$ may change unpredictably compared to the value for the current $x$.

Two examples are now considered, which showcase the aforementioned effects of step size choice on total error. In each example, two different step sizes are tested, both of which are close to the optimal value. The $x$ value is varied about its nominal value, and the relative error in $df/dx$ (as compared to the true

derivative) is plotted using each test step size. In addition, a line is drawn for each step size that indicates the true truncation error $TE(x, h)$ for the nominal $x$ value. This line is intended to provide a quick visual indication of the accuracy of the estimated truncation error.

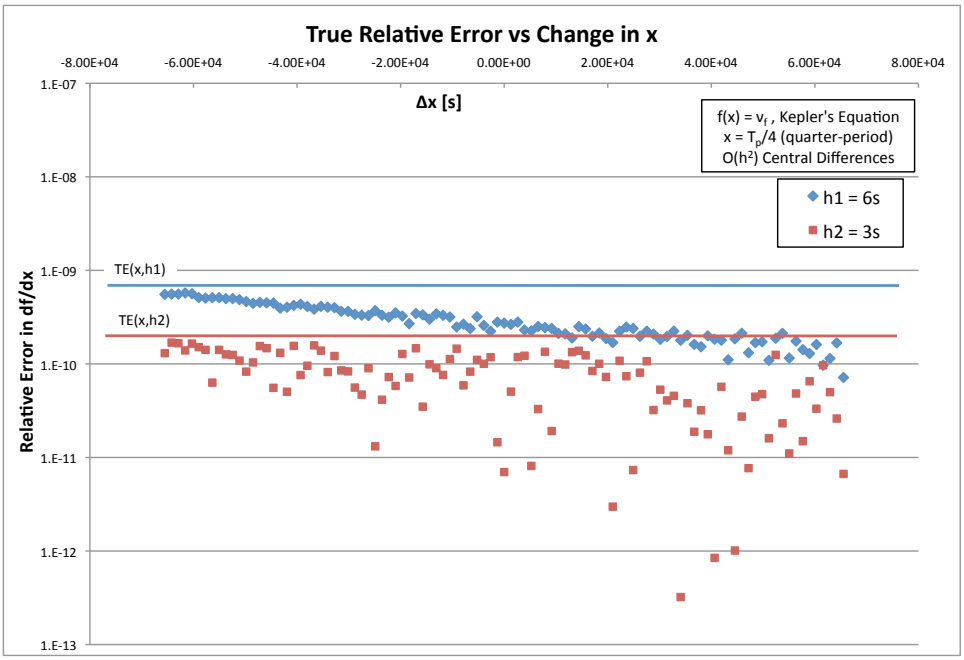


Figure 3.4: The effects of near-optimal step sizes.

The first example replicates the Kepler propagation problem given in Table 3.2, using step sizes of $h_1 = 6[s]$ and $h_2 = 3[s]$. The nominal $x$ (where $x = t_f$) is varied by $|\Delta x| \leq 65536[s]$, and the results are given in Figure 3.4. As expected, the larger step size results in greater relative error for almost all tested values of $x$. It is also clear that the truncation error estimates at the nominal $x$ do in fact bound their respective relative errors for the range of $x$ values tested. This is no coincidence, and is explained in Section 3.5.3. Finally, the drastic reduction in

the spread of relative error points for the $h_2$ scan at the negative $\Delta x$ values is an indication that there may be a better optimal step size as $x$ reduces towards zero. This is a graphical confirmation of the commonly known trend that if the magnitude of $x$ is greatly reduced, the optimal step size will follow suit.

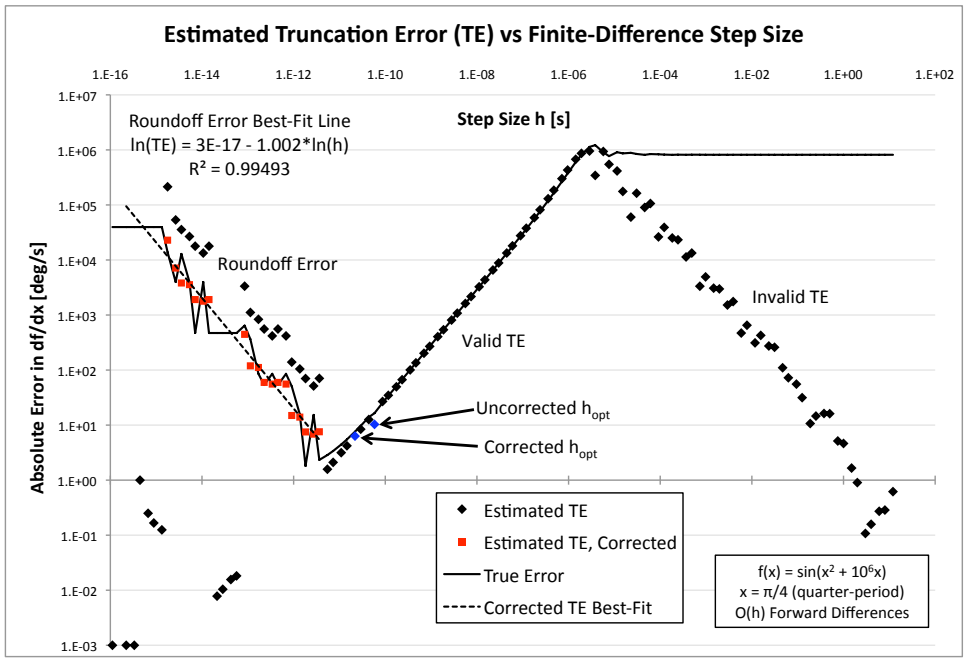


Figure 3.5: The simple algorithm applied to an ill-conditioned function.

The second example revisits the ill-conditioned function $f(x) = sin(x^2 + 10^6 x)$ at $x = \pi/4$, originally analyzed in Figures 2.5 and 2.8. The corrected and uncorrected optimal step sizes are indicated in Figure 3.5. It is apparent that, as expected, the truncation error best-fit line very closely approximates the corrected and true roundoff errors. The uncorrected and corrected optimal step sizes are $h_{\text{opt}} \approx 5.82e{-}11$ and $h_{\text{opt}} \approx 2.18e{-}11$, respectively. In this instance, a visual inspection suggests that the corrected optimal step size may still be slightly

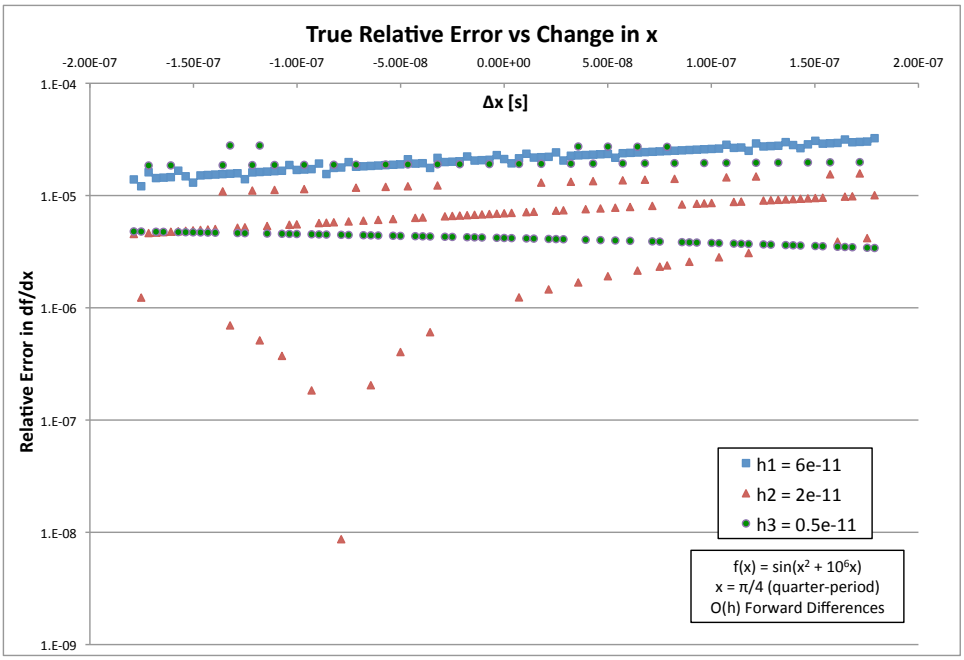


Figure 3.6: Increase in true relative error with a continued decrease in step size.

higher than the true optimal step size. Figure 3.6 shows the effects of varying the nominal $x$ by $|\Delta x| \leq 1.75e{-}8$, by plotting the true relative error in the FDD value (as compared to the true derivative). Three successively decreasing step sizes are chosen to highlight the effects of excessively reducing the step size on FDD values at neighboring $x$ points. The largest step size $h_1$ is clearly greater than the optimal step size (from Figure 3.5), and as expected its associated relative error changes fairly predictably with $x$. The next step size $h_2$, being much closer to the optimal value, has less relative error but the actual values are much less predictable with respect to changes in $x$. As in the previous example, this is expected because although the optimal step size results in less total error, it is also much more susceptible to roundoff errors which affect the total error in unpredictable ways

(on a small scale). However, this trend is partially broken in the smallest step size $h_3$, for which the relative error increases greatly and the uncertainty in the relative error is also high. In other words, the step size $h_3$ provides neither the smaller errors of $h_2$ nor the more predictable errors of $h_1$; it is considered to be a 'worse' choice than both.

The results of these examples are indicative of the general trend when the chosen step size is in the vicinity of the optimal step size. For step sizes slightly larger than the optimal, the FDD error for neighboring $x$ values has high precision[2] but low accuracy[3]. As the step size approaches its optimal value, the computed FDD error reduces in precision, but increases in accuracy. Finally, as the step size is further reduced, it loses both accuracy and precision. These observations are backed by theory as explained at the beginning of this section.

One of the most common reasons for using a FDD method is to compute a derivative for use within an optimization loop. Because this loop may involve many iterations, it is likely desirable to compute the optimal step size once, and use that step size as iterations progress and the optimization parameter $x$ is changed. This is in fact the exact situation considered in this section. Therefore a step-size smaller than the optimal should be avoided at all costs, since it provides neither precision nor accuracy in the computed FDD values when $x$ is varied. This leaves only a small range of useful step sizes, all greater than or equal to the optimal, which present a design tradeoff between precision (larger step sizes) and accuracy

[2]Precision indicates how close the FDD values are to each other.
[3]Accuracy indicates how close the average FDD value is to the true derivative.

(smaller step sizes).

The deviation tolerance employed by the simple step-size search algorithm from Section 3.5.1 encompasses this design tradeoff. A tight tolerance causes the algorithm to finish early with larger step size values, while a loose tolerance allows the algorithm to proceed closer to the optimal step size. However, a tolerance that is too loose may result in the algorithm proceeding past the optimal step size, which as stated before is always undesirable. It has been found in practice that a deviation tolerance equal to the desired truncation error slope $n$ produces optimal step sizes which, after correction, provide a very good balance between precision and accuracy. Such a tolerance effectively stops the algorithm as soon as a truncation error value is greater than the value preceding it, as evidenced in the uncorrected optimal step sizes computed in Figures 3.3 and 3.5. However, this rule of thumb does not have to be followed; the deviation tolerance can be left as a user-specified parameter if it is so desired.

### 3.5.3 Optimal Step Size Validity Range

The step size computed by the simple algorithm of Section 3.5.1 and subject to the constraints of Section 3.5.2 results in accurate and precise FDD values for the current and neighboring values of the independent variable $x$. A method is now derived to compute the particular range of $x$ values for which the optimal step size is valid.

General FDD methods are derived by first fitting an $p^{th}$-degree Taylor polynomial (where $p = n + d - 1$ from Section 2.4) to a function $f(x)$ at $p + 1$

points in the neighborhood of a particular $x_0$. This polynomial is differentiated $d$ times at $x$, and the result is taken to be an approximation of the true derivative $f^{(d)}(x)$. The error $R_n$ in this approximation, proportional to $O(h^n)$ and given in Lagrange form in Section 2.3, is rewritten here

$$R_n = a_1 f^{(n+d)}(\xi) h^n \ , \ \xi \in [x_0 - a_2 h, x_0 + a_3 h] \tag{3.10}$$

where $a_1$, $a_2$, and $a_3$ are known constants dependent upon the particular FDD method used, $\xi$ is an unknown parameter, and the step size $h$ is the distance between the extremal polynomial fit points. It was shown in Section 2.6 that the coefficient $f^{(n+d)}(\xi)$ can be estimated by using two step sizes $h_1$ and $h_2$. Note that estimating $f^{(n+d)}(\xi)$ is tantamount to estimating $\xi$ itself, although an actual value for $\xi$ is not of interest.

Consider the case where $f^{(n+d)}(\xi)$ is estimated twice, first using step sizes $h_1$ and $h_2$, and then using $h_2$ and $h_3$, and the estimates are equal to each other. This can only happen for one of two reasons: either $\xi$ is the same for both estimates, or $\xi$ changed but $f^{(n+d)}(\xi)$ happened to be the same in both cases. Both of these cases are purely coincidental; they can only occur for particular combinations of $f$, $x_0$, and $h$. Figure 3.7 illustrates these caes when the first derivative is estimated ($d = 1$).

Consider next the case where *many* different step sizes in the range $[h_{\text{opt}}, h_{\text{max}}]$ are tested (where $h_{\text{max}} \gg h_{\text{opt}}$), and $f^{(n+d)}(\xi)$ is the same for all of them. Extending the reasoning from above, this can only occur for one of two reasons:

1. $\xi$ is the same for all step sizes. Because the range of possible $\xi$ values depends

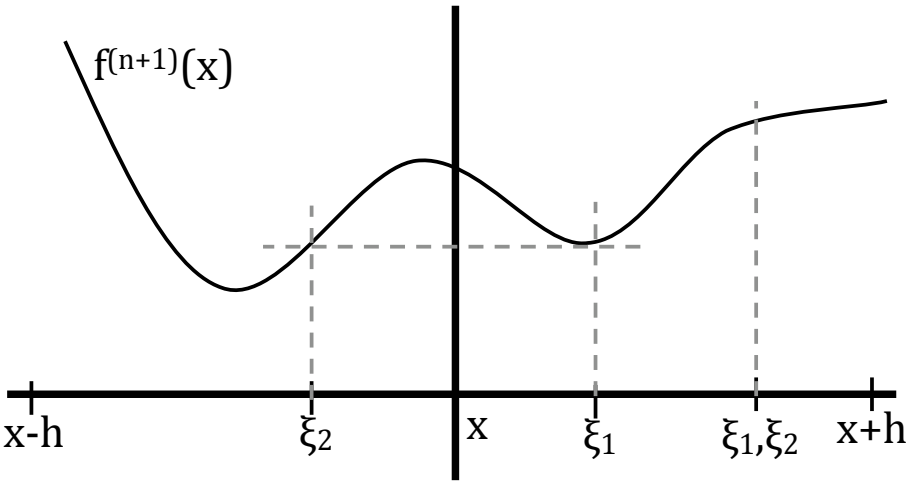


Figure 3.7: Two possibilities of $\xi_1$ and $\xi_2$ for which $f^{(n+1)}(\xi_1) = f^{(n+1)}(\xi_2)$.

on the step size, the $\xi_{\text{opt}}$ associated with $h_{\text{opt}}$ will be very close to $x$ itself. As a consequence, all of the $\xi$ values must be very close to $x$. This presents a contradiction; a general $\xi$ value need not be close to $x$ if the associated step size $h$ is large. Therefore this possibility is precluded.

2. $f^{(n+d)}(x)$ repeats itself for all $\xi$ values. Because the $\xi$ values are not necessarily evenly spaced, the possibility of $f^{(n+d)}$ coincidentally repeating itself many times at the exact $\xi$ values is negligible. On the other hand, if $f^{(n+d)}$ is constant over the range $[x - a_2 h_{\max}, x + a_3 h_{\max}]$ then this case would also be true.

By elimination, it is shown that if $f^{(n+d)}(\xi)$ is constant for many tested step sizes in the range $[h_{\text{opt}}, h_{\max}]$, then $f^{(n+d)}(x)$ must in fact be constant for all $x \in [x_0 - a_2 h_{\max}, x_0 + a_3 h_{\max}]$. An example is presented to show how this conclusion is used along with the simple step-size algorithm to determine $h_{\max}$.

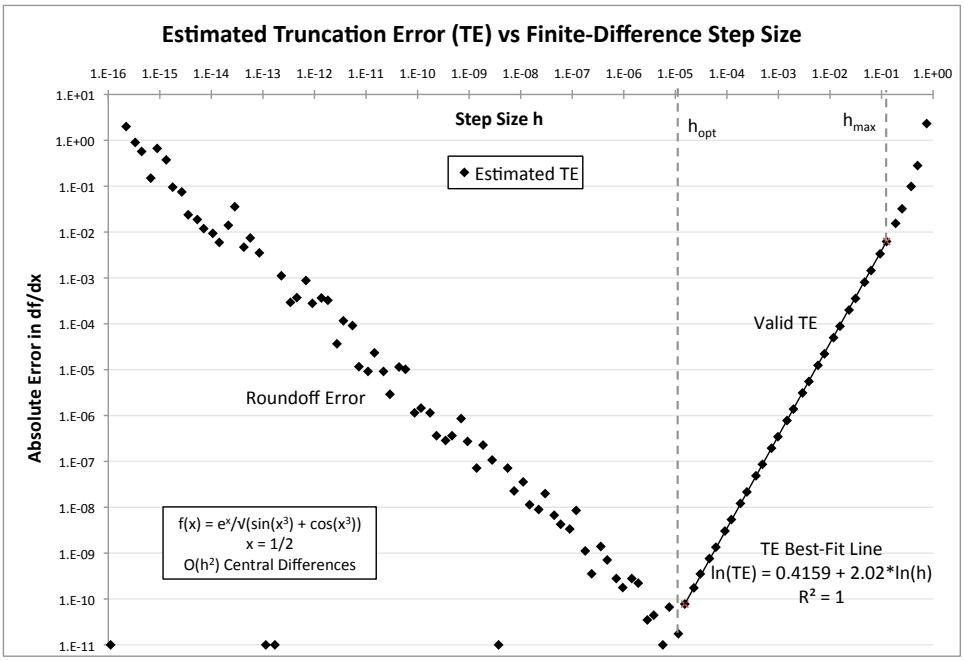


Figure 3.8: Determination of a maximum step size $h_{\max}$.

**Example 3.3.** Figure 3.8 shows the estimated truncation error for the function

$$f(x) = \frac{e^x}{\sqrt{sin(x^3) + cos(x^3)}} \tag{3.11}$$

which was also evaluated by Lyness [29], Squire and Trapp [50], and Pemba [37]. After evaluating this function with the simple step-size algorithm, it is found that the uncorrected optimal step size $h_{\text{opt}}$ is approximately $1.14e{-}5$. At the initial large step sizes near unity, the slope is deemed inaccurate as compared to the expected slope of $n = 2$. However, it is determined that for step sizes smaller than 0.125, the truncation error estimate slope is within a tight tolerance of the expected slope. For comparison, the truncation error equation for a $2^{nd}$-order

central-difference FDD method, on a log-log scale, is

$$\ln TE_2 = \ln(f^{(3)}(\xi)/6) + 2\ln h \tag{3.12}$$

In this example it is found that for $h \in [h_{\text{opt}}, h_{\text{max}}]$ (where $h_{\text{max}} = 0.125$), the slope is very close to the expected value, as confirmed by the best-fit line in Figure 3.8. This means that over the given step size range, the derivative $f^{(n+d)} = f^{(3)}$ is nearly constant for all tested step sizes. By the reasoning presented prior to this example, a near-constant $(n+d)^{th}$ derivative is expected over the range $x \in [0.5 - 0.125, 0.5 + 0.125]$.

If it is shown that the $(n+d)^{th}$ derivative is nearly constant over a specified range of $x$ values, then the error in the $n$-th order FDD approximation to the derivative is expected to be nearly constant over the same range of $x$ values. From a different perspective, this is equivalent to noting that the difference between using $p$-th and $(p+1)$-th degree Taylor polynomials to fit the function $f(x)$ at a given $x$ is nearly constant for step sizes in the range $h \in [h_{\text{opt}}, h_{\text{max}}]$.

The proofs given and conclusions reached so far in this section are now combined for clarity. When the simple step-size search algorithm is initialized with a step size $h_0$ which is too large, it will iteratively reduce this step size until the truncation error slope reaches the expected value. The first step size for which this is true is $h_{\text{max}}$, as proven above. Continuing onwards, the algorithm determines the minimum safe step size $h_{\text{opt}}$, which can be corrected with (2.83) if desired. If this algorithm is used within an optimization loop, then there is no need to recompute $h_{\text{opt}}$ for future iterations of $x \to x'$; the same $h_{\text{opt}}$ can be used

while $x' \in [x - a_2 h_{\max}, x + a_3 h_{\max}]$, and the resulting estimated truncation error will be consistent with the error for $x$.[4]

As mentioned in Section 3.5.2, it is no coincidence that the true relative error in Figure 3.4 remains less than the predicted relative truncation error. In fact, the range of $x$ values used are based on $h_{\max}$ as computed using the simple step-size search algorithm.

## 3.6 Deviant Functions

Certain families of functions have behavior that falls outside the assumptions made in the theoretical analysis of Chapter 2, the numerical analysis of the current chapter, and the simple step-size search algorithm of Section 3.5.1. These functions can cause the simple search algorithm to fail, so it is necessary to identify and account for them by appropriately modifying the search algorithm.

### 3.6.1 Low-Degree Polynomial Functions

If an $n^{th}$-order FDD method is used to approximate the $d^{th}$ derivative of a polynomial with degree less than $(n+d)$, then the approximation will be exact, and there will be no truncation error. This is easily seen by the fact that truncation error for such a FDD method is,

$$TE_n(x, h) = a_1 f^{(n+d)}(\xi) h^n \tag{3.13}$$

[4] $a_2$ and $a_3$ are nonzero constants, defined in (2.29) and determined by the particular FDD method.

where $f^{(n+d)} = 0$ for polynomials of degree $< (n+d)$. If the simple step-size search algorithm is applied to such a function, and power-of-2 step sizes are used, then all truncation error estimates will be zero. Detecting such cases is not difficult; if the first few truncation errors are zero, then it can be assumed that the function is a low-degree polynomial. Choosing which step size should be returned by the algorithm, however, is slightly more ambiguous since all step sizes produce an exact derivative for all $x$ values. Therefore, without any loss of generality or accuracy, the optimal step size $h_{\text{opt}}$ is chosen to be the smallest tested step size, and the maximum valid step size $h_{\max}$ is chosen to be equal to the initial trial step size. Note that it is unsafe to choose $h_{\max}$ any larger than this, because the function's behavior is unknown outside of the domain $[x - a_2 h_{\max}, x + a_3 h_{\max}]$.

### 3.6.2 Functions with Null High-Order Derivatives

Certain functions have generally nonzero derivatives, with the exception of one or more high-order derivatives which happen to be zero at the point of differentiation $x$. If an $n^{th}$-order FDD method is used, and the $(n+d)^{th}$ derivative is zero at $x$, then the slope of the truncation error estimate will not have the expected value of $n$. An example is the simple polynomial,

$$f(x) = \frac{x^5}{60} - \frac{x^3}{6} \tag{3.14}$$

Figure 3.9 shows the estimated truncation errors when the $O(h^2)$ central-difference method is used to analyze this function at $x = 1$. The simple step-size algorithm, when applied to this function, expects a truncation error slope of $n = 2$. However,

it is clear from Figure 3.9 that this slope is not reached; the slope in the valid truncation error region is actually 3.9917.

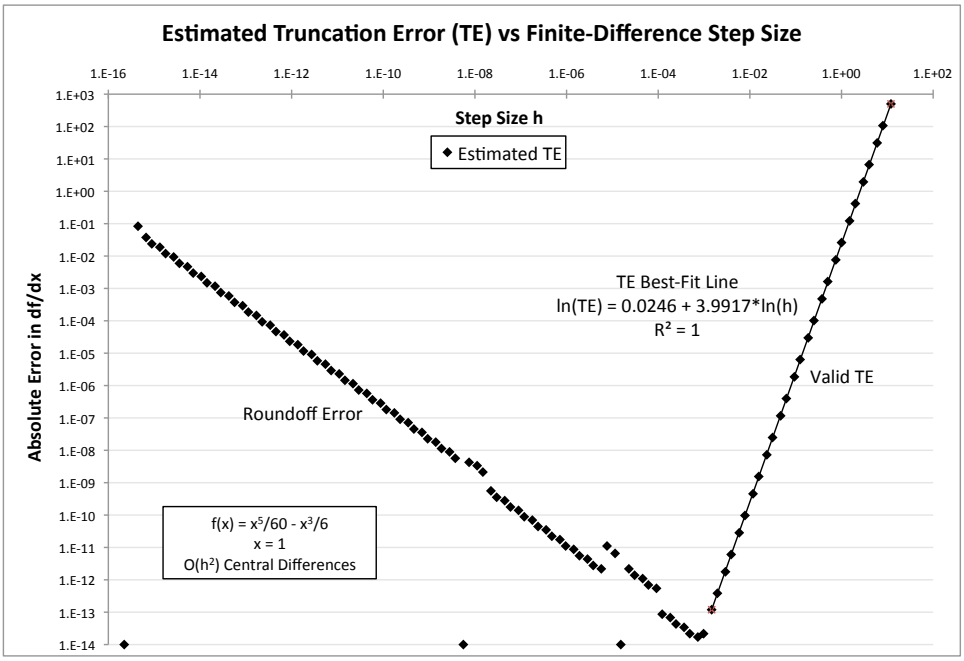


Figure 3.9: Unexpected truncation error slope for a deviant function.

To understand the reasoning behind this behavior, a higher-order derivative of $f(x)$ must be considered,

$$f^{(3)}(x) = x^2 - 1 \tag{3.15}$$

At the point of differentiation $x = 1$, this derivative is in fact zero. Writing the central-difference method to a higher-order error using the Lagrange remainder,

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(x)}{3!}h^2 - \frac{f^{(5)}(\xi)}{5!}h^4 \tag{3.16}$$

where again, $\xi \in [x-h, x+h]$. Applied to the current function, the $3^{rd}$-order

derivative is zero, and the FDD equation reduces to,

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(5)}(\xi)}{5!}h^4 \tag{3.17}$$

In this case, the total error function (2.62), when analyzed from a log-log perspective, is

$$E(x,h) = \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h} + |C_4|h^4 \tag{3.18}$$

$$\ln E(x,h) = \begin{cases} \ln\left(\epsilon|F_\epsilon| + \delta|F_\delta|\right) - \ln h & h \ll h_{\text{opt}} \\ \\ \ln|C_4| + 4\ln h & h \gg h_{\text{opt}} \end{cases} \tag{3.19}$$

It is now apparent that, for step sizes larger than the optimal step size, the expected log-log slope of the total error is 4, which confirms the observed truncation error slope in Figure 3.9.

This problem can be generalized as follows. For a function $f(x)$, which is differentiated $d$ times at $x$ using a particular FDD method of order $n$, the truncation error of the FDD method can be expressed as

$$TE_n(x,h) = C_n h^n + C_{2n}h^{2n} + C_{3n}h^{3n} + C_{4n}h^{4n} + \cdots \tag{3.20}$$

where each $C_{jn}$ is proportional to $f^{(jn+d)}(x)$. If $f^{(n+d)}$ is zero at the given $x$, then $C_n = 0$ and the truncation error slope must be analyzed using the $C_{2n}$ term. Similarly, if $f^{(2n+d)}$ is also zero at the given $x$, then $C_{2n} = C_n = 0$ and analysis must be done with the $C_{4n}$ term. In general, the truncation error slope is analyzed using the smallest $j$ such that $C_{jn}$ is nonzero, which results in an expected log-log slope of $jn$ itself.

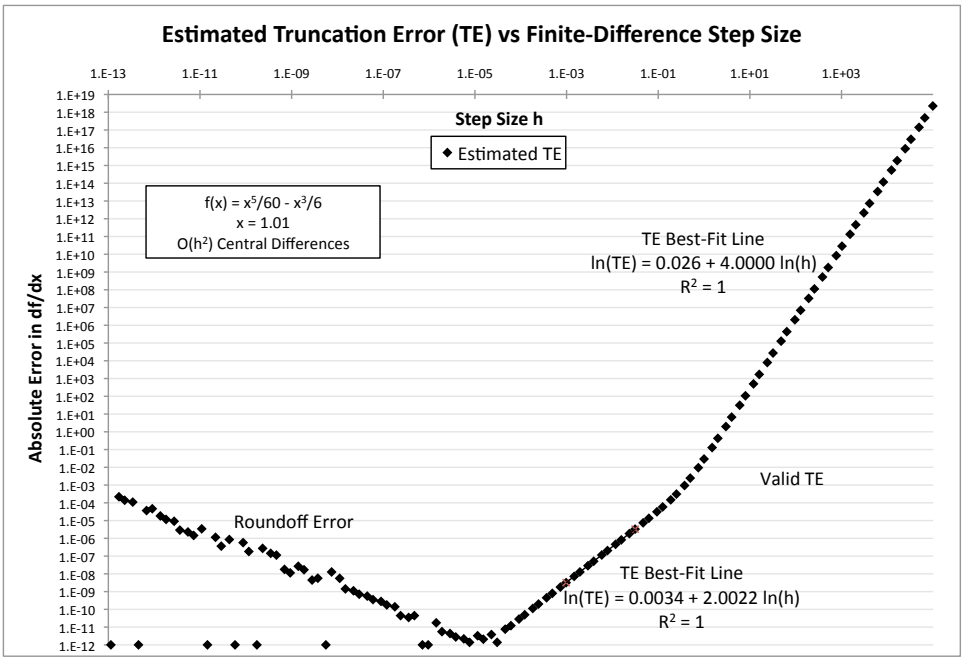


Figure 3.10: Unexpected truncation error slope resolving to the expected slope.

Even this search method can get stuck if there is a $C_{jn}$ which is very close to (but not equal to) zero. For example, when the same $5^{th}$-order polynomial is analyzed at $x = 1.01$, the derivative $f^{(3)}(x)$ is nearly zero. Figure 3.10 shows that the truncation error slope is $n = 4$ for large step sizes, as expected by the above analysis. However, because $f^{(3)}(x)$ is in fact nonzero, for small step sizes the truncation error slope returns to the expected value of $n = 2$ given by the FDD order itself.

To account for these unexpected truncation error slopes, the simple step-size search algorithm must be willing to accept slopes other than $n$. Instead of assuming that the expected truncation error slope is $n$, the algorithm should look

for a slope of $jn$ where $j$ is a positive integer. This slope is followed down a decreasing sequence of step sizes, until the slope changes. If the new slope is $j'n$ where $j'$ is an integer such that $1 <= j' < j$, then the estimated truncation errors are still valid. However, if the new slope does not fall into this range, then it is assumed that the current step size has reached the domain of roundoff error.

### 3.6.3 Functions with Null Odd- or Even-Order Derivatives

Trigonometric functions have derivatives which repeat on a regular basis. For example, the function

$$f(x) = \sin(x)\cos(x) \tag{3.21}$$

repeats every second derivative,

$$f^{(d)}(x) = \begin{cases} C_1 \sin(x)\cos(x) & \text{d even} \\ C_2 \cos(2x) & \text{d odd} \end{cases} \tag{3.22}$$

where $C_1$ and $C_2$ are constants associated with the derivative order $d$. If the first derivative ($d = 1$) is evaluated at $x = \pi/4$ using a $2^{nd}$-order FDD approximation ($n = 2$), then all odd derivatives are zero. This causes the truncation error to be zero (for the FDD used).

Figure 3.11 shows the result of using the simple step-size search algorithm on this function. It can be seen that, as predicted, the estimated truncation error is within machine precision of zero for large step sizes. Because the algorithm searches for a truncation error slope of $jn$, and no such slope exists for this function, the algorithm will continue to search until it reaches a minimum allowable step size. A visual examination of Figure 3.11 indicates that the 'best' step size

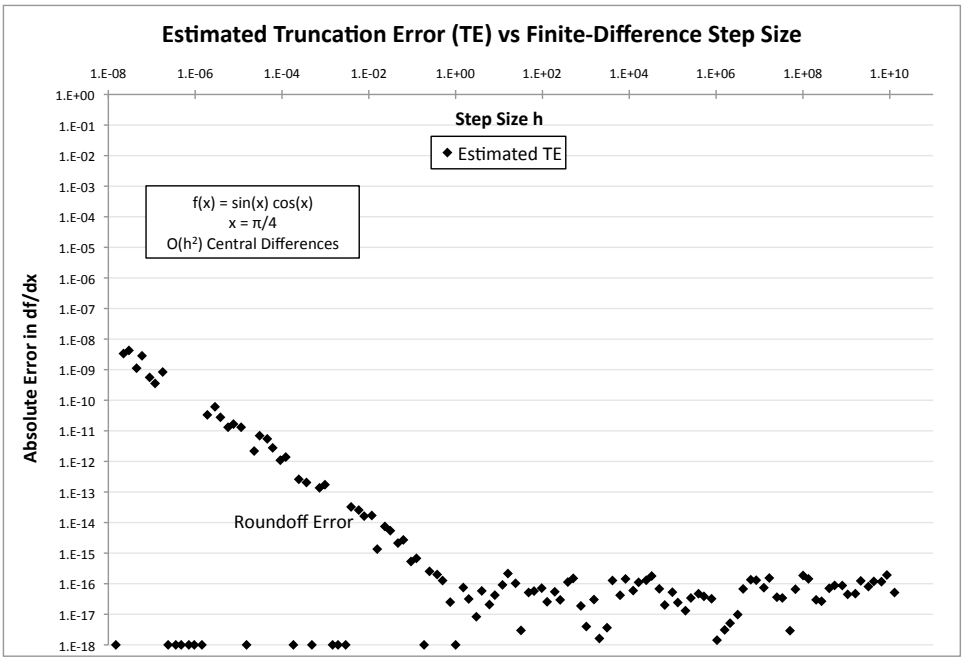


Figure 3.11: Function whose odd derivatives are all zero at a particular $x$.

would be near $h = 1$, which is the minimum step size for which roundoff error has not yet crept in. However, the current iterative form of the search algorithm cannot come to this answer.

It should be noted that this is not a critical drawback in the algorithm. Situations such as this have only been observed by the author for certain trigonometric functions evaluated at specific $x$ values. In these cases, the algorithm can simply return the initial tested step size as $h_{\text{opt}}$, and set $h_{\max} = 0$ to indicate that the function should be reanalyzed as soon as $x$ changes.

## 3.7 Extension to Multidimensional Functions

More often than not, the function being differentiated is a vector function of a vector input. In this case, the Jacobian matrix of the function with respect to the input is necessary for gradient-based optimization techniques. When FDD methods are used to compute the Jacobian's constituent gradient vectors, a single step size is usually used for each input variable. However, there is no guarantee that a given step size will be optimal for every component of the output vector. Because of this, it is of interest to consider the derivative of each output component with respect to each input variable separately.

The simple step-size search algorithm could certainly be used for a given input variable, and directed towards analyzing the truncation error for only a particular output variable. This algorithm would then be run from within a nested loop, with the outer loop iterating over the $n$ input variables, and the inner loop iterating over the $m$ output variables. Such an analysis method has a cost of $O(nm)$, but this is easily reduced by noting that the function generally computes *all* outputs for any given set of inputs. With this in mind, it is straightforward to modify the search algorithm by having it analyze the estimated truncation error for each element of the output vector independently, within an internal loop. Not only does this reduce the algorithm's functional cost to $O(n)$, but it also noticeably cuts down on the performance and memory overhead involved with more frequent calls of the algorithm.

Consider a modified form of the orbit propagation problem from Section 3.5.1, specified in Table 3.2. The orbit is now propagated for half its period,

and the output is the position vector $\mathbf{r}_f$ at the final time $t_f$. The gradient of the function is therefore the final velocity vector $\mathbf{v}_f$, with the true value being computed via a Kepler propagation and standard coordinate transformations.

Table 3.4: Solution for multidimensional Kepler orbit propagation problem.

| | $\mathbf{r}_{f,x}$ | $\mathbf{r}_{f,y}$ | $\mathbf{r}_{f,z}$ |
|---|---|---|---|
| $h_{\text{opt}}$ [s] | 4.0 | 6.0 | 6.0 |
| Relative Error | $2.269e{-10}$ | $1.373e{-10}$ | $1.400e{-10}$ |
| Calls to f() | 131 | 129 | 129 |

Table 3.4 gives the uncorrected optimal step sizes[5] for each component of the gradient $d\mathbf{r}_f/dt_f$, along with the relative error (with respect to the true derivative) and the number of function calls required to compute each derivative separately[6]. The total number of function calls would therefore be the sum of the individual values (in this case, 389). In contrast, if the multidimensional version of the algorithm is used, then the total number of function calls will be the maximum of the individual values.

### 3.7.1 Choosing a Single Optimal Step Size

When there are multiple optimal step sizes for a given input variable, it is important to be able to choose one of these for use in future iterations of an optimization loop. In the multidimensional orbit propagation problem of Table

[5]It is proven in Section 2.9 that the corrected optimal step size is easily computed from the uncorrected value.

[6]The initial trial step size is taken to be 10000 times greater than $x$ itself, which is responsible for the large number of function evaluations.

3.4, the optimal step sizes for each output element were shown to be quite close to each other so choosing between them is not a difficult choice. This is not necessarily true (e.g. ill-conditioned functions), and so it becomes necessary to study various methods to choose the most desirable optimal step size for a given problem. It is empirically determined that three straightforward methods are min(), max(), and mean(), as described below.

1. min($h_{\text{opt,i}}$): The smallest of all possible optimal step sizes is chosen. This method is most useful when most of the $h_{\text{opt}}$ values are close to the minimum (in magnitude), with a few exceptions. For the output elements which require a large step size, roundoff error will cause the total error to increase in proportion to the order of the derivative $d$ being sought.

2. max($h_{\text{opt,i}}$): The largest of all possible optimal step sizes is chosen. This is a direct contrast to the min() method, and is appropriate when most $h_{\text{opt}}$ values are close to the maximum. In this case, the total error for output elements requiring a small step size will increase in proportion to the order of the FDD method being used. Because the order of the FDD method $n$ is generally larger than the order of the derivative $d$ being sought, it is rare for the max() option to be the best.

3. mean($h_{\text{opt,i}}$): The log-based mean of all $h_{\text{opt}}$ values is chosen as the optimal value, causing the increased error to be distributed amongst all derivatives. When the $n > d$, it is desirable to weight the computed $h_{\text{opt}}$ towards the smaller step sizes. The increased roundoff error for some derivatives will

be outweighed by the decreased truncation error of the rest. While the thorough weighting method would be to solve a polynomial for $h_{\text{opt}}$ in terms of all other $h_{\text{opt,i}}$ values, in practice it is much easier to simply use a standard $n$-vs-$d$ weighting scheme,

$$\ln(h_{\text{opt}}) = \ln(h_{\text{opt,min}}) + \frac{d}{n+d}(\ln(h_{\text{opt,max}}) - \ln(h_{\text{opt,min}})) \tag{3.23}$$

Regardless of the method used to choose an $h_{\text{opt}}$ value, an analyst should be vigilant of situations where the computed optimal step size for a given input varies greatly (in magnitude) for the various function outputs. This usually implies that some components of the function are ill-conditioned as compared to others. Such a situation is known to present difficulties for numerical optimization techniques.

## 3.8 Chapter Conclusions

A numerical analysis of the step-size theory from Chapter 2 was performed in this chapter. It was first shown that step sizes which are powers-of-2 should be used. Doing so eliminates a particular form of representation error from the computed FDD values, and consequently from the estimated truncation error values.

Next, it was shown that when the estimated truncation error (as a function of the step size) is analyzed with log-log scaling, its slope is predictable. In particular, when truncation error is valid, the slope is equal to the order of the FDD method $n$. When roundoff error has dominated the results, the best-fit line of the slope is the negative of the derivative order $d$. Finally, for very large step

sizes for which neither roundoff nor truncation errors are valid, it was shown that the slope of the estimated truncation error is unpredictable.

The knowledge of estimated truncation error slope was used to develop a simple step-size search algorithm. This algorithm, when initialized with a very large step size $h_0$, is capable of 'skipping over' the initial incorrect FDD values. The region of valid truncation error is successfully sought out by its expected slope, and the step size is further reduced until roundoff error begins to dominate the error. The step size at this point is considered as the uncorrected optimal step size.

When this optimal step size is used with neighboring values of the independent variable, is was shown that the optimal step size gives a good balance between precision and accuracy of the resulting derivatives. While increasing the step size increases precision at the expense of accuracy, it was shown that further decreasing the step size below the optimal value results in decreased precision and accuracy.

In recognition of the fact that FDD methods are often used within an optimization loop, it was shown that there is a computable range of independent variable values for which the optimal step size produces a predictable amount of error. This is one level of optimization; the search algorithm can be called once to determine $h_{\text{opt}}$, and then not called again until the independent variable changes by a predetermined amount.

It was shown that certain families of functions can trip up the simple step-

size algorithm. These families of functions were analyzed and it was determined that the algorithm could be modified to account for them. There is no loss of generality in the algorithm from these modifications.

Finally, the algorithm was extended to account for multidimensional functions of a vector input. It was shown that, with appropriate considerations, the runtime expense can be limited to $O(n)$ where $n$ is the size of the input vector.

For multidimensional functions, there is a possibility that the optimal step size is different for each component of the output. Choosing one of these step sizes presents a design tradeoff. Three different methods were presented of choosing between the various optimal step sizes.

# Chapter 4

# Numerical Examples

## 4.1 Chapter Summary

The step-size analysis theory and algorithm from Chapters 2 and 3 are tested in this chapter. A particular implementation of the simple step-size search algorithm, called AutoDX, is used to determine the optimal step size for a variety of functions. Each test case is chosen to showcase a different facet of the theory and algorithm, from the basic goal of determining an optimal step size to more advanced fault-tolerance capabilities.

For each test case, the results of the AutoDX algorithm are compared to those obtained using two other methods. The first comparison method involves using a rule-of-thumb step size. The second method, called the Gradient Tuned Algorithm (GTA), is a statistical step-size search algorithm created by Ocampo and Restrepo [42]. The GTA algorithm starts with a very small step size (e.g. $h_0 \approx 10^{-16}(1 + |x|)$), and analyzes 'batches' of FDD values at each step size up to some maximum. For each batch of FDD values, the algorithm computes statistical quantities such as the mean and dispersion (standard deviation). The idea behind GTA is that for very small step sizes, the dispersion of FDD values will be high due to large roundoff errors. As the step size increases (i.e. approaches

$h_{\text{opt}}$ from the roundoff error side) the dispersion should decrease due to decreasing roundoff error. This trend continues until the optimal step size is reached, after which truncation error dominates and the dispersion increases again. The GTA algorithm then returns the step size with least FDD dispersion as the optimal step size.

It should be noted that in many optimization algorithms, the step size used to compute a FDD value is itself computed from a relative epsilon (*eps*) value, where

$$h = eps(1 + |x|) \tag{4.1}$$

While the GTA subroutine returns $eps_{opt}$ for the optimal step size, the AutoDX subroutine instead directly returns $h_{\text{opt}}$. When comparing results, the GTA *eps* is converted to the equivalent step size. However, when examining the effects of the optimal step size on a range of $x$ values, the AutoDX $h_{\text{opt}}$ is first converted to its equivalent *eps*, which is then used for all tested $x$ values. This is done to more accurately reflect real-world usage of FDD methods within an optimization loop.

## 4.2 Examples of Fundamental Functions

Fundamental functions compute their output by applying primitive operators (i.e. add, subtract, multiply, divide) to built-in functions. Examples are polynomial, trigonometric, and exponential functions, or combinations thereof. The output of such a function is expected to have little to no condition error, since built-in functions and primitive operators are accurate to machine precision.

**Example 4.1.** The function

$$f(x) = x^2 + x - 1.34 \tag{4.2}$$

is differentiated at $x = 3.1$, using the $O(h^2)$ central-differences method. Here, $n = 2$ and $d = 1$. For the AutoDX algorithm, the initial large step size is taken to be $h_0 = 1e5(1 + |x|)$. The rule-of-thumb step size is $h_{\text{rt}} = 5e{-}6|x|$, as per the guideline $h_{\text{rt}} = \delta^{1/3}|x|$ for second-order central-difference methods. The solution using the three comparison algorithms is given in Table 4.1. The true relative error is referenced to the true derivative, and the estimated relative error comes from the total error function E(x,h) in (2.62). The AutoDX algorithm determined that the maximum valid step size $h_{\max} = 192$, and that the condition error in $f(x)$ is less than machine precision. Note that the estimated error is consistently larger than the true error; this is because the total error function is by design an upper bound to the true error.

Table 4.1: Solution for $df/dx$ from example 4.1.

| | $h_{\text{opt}}$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | $1.55e{-}5$ | $1.99e{-}11$ | $1.23e{-}9$ | 2 |
| AutoDX | 32.0 | $1.23e{-}16$ | $6.10e{-}16$ | 55 |
| GTA | $4.51e{-}6$ | 0.00 | $4.33e{-}9$ | 144 |

Clearly the computed derivative should be exact for any reasonable step size, since this FDD method has no truncation error for polynomials of degree $\leq 2$. However, the results for this example show the far-reaching effects of numerical errors. The relative error is zero for the GTA results, and near zero for the

AutoDX results, even though the respective computed optimal step sizes are vastly different. Meanwhile, the optimal step sizes between GTA and the rule-of-thumb method are very similar but their relative errors are not.

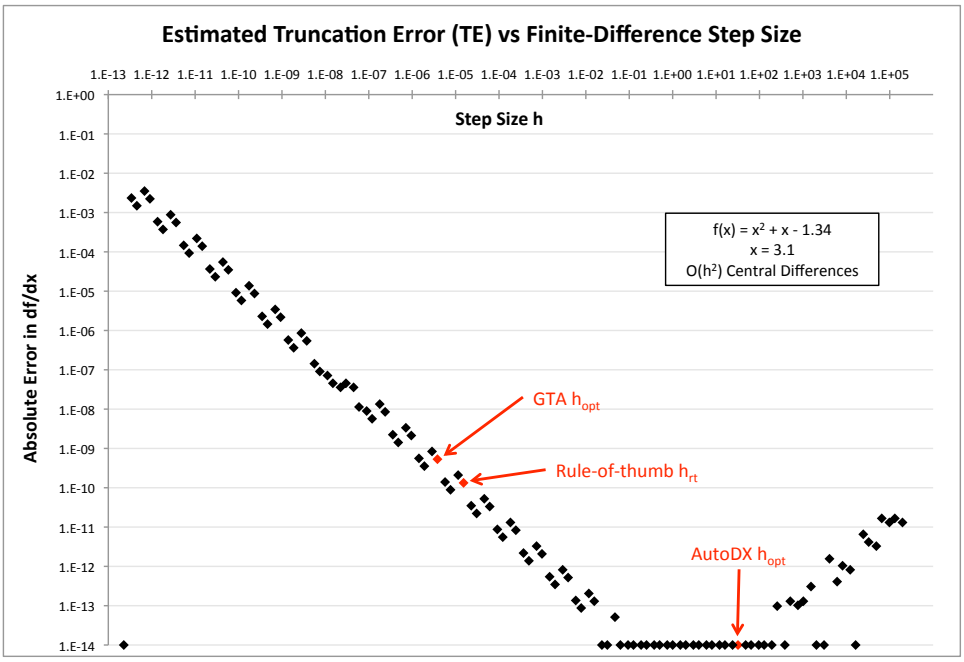


Figure 4.1: Estimated truncation errors for example 4.1.

This conundrum is better understood by looking at the estimated truncation error plot given in Figure 4.1[1]. Because truncation error is nonexistent, only numerical sources can cause nonzero estimated errors. In fact, it is seen that there is only a small step-size range for which the estimated error is zero. The AutoDX algorithm noticed this during its search, and immediately recognized the low-

[1]The true error is omitted from here on out, because the true derivative is assumed to be unknown. The optimal step size can easily be corrected to represent the true optimal step size using (2.83).

degree polynomial exception from Section 3.6.1. The GTA algorithm, meanwhile, found a range of step sizes for which the FDD dispersion was locally minimized.

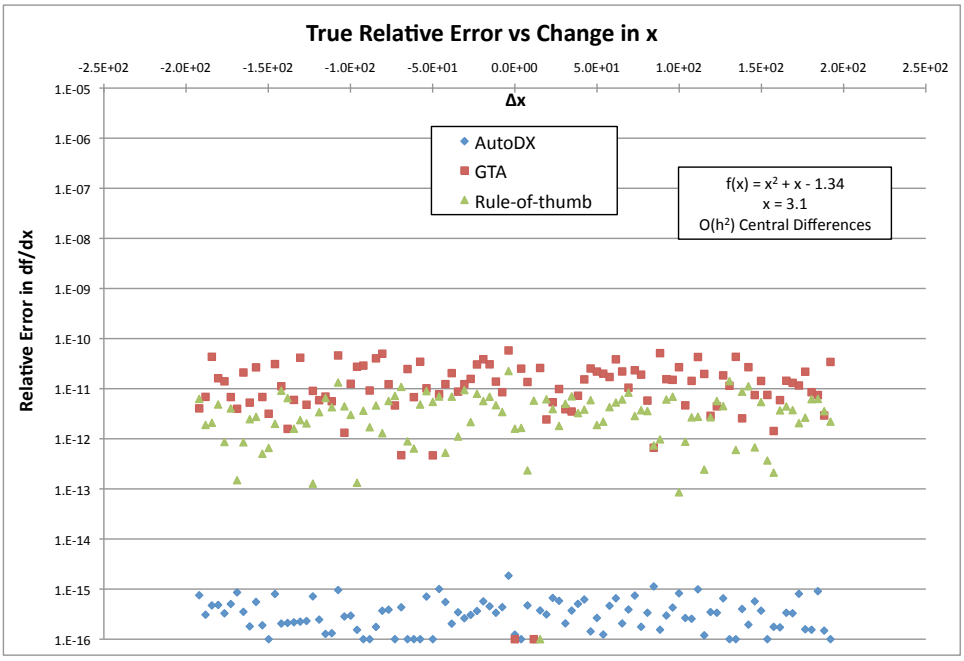


Figure 4.2: Relative errors with respect to neighboring $x$ values for example 4.1.

The numerical problem in this case occurs because of representation error in $x$. Even though power-of-2 step sizes are chosen (as explained in Section 3.3), $x$ itself is not exactly represented[2] and so there are minor errors in $x \pm h$. As a consequence, step sizes much smaller than 0.1 (the approximate step size from Figure 4.1 at which roundoff error begins to dominate) are predicted to have an increasing amount of error. This is confirmed in the error of the rule-of-thumb step size $h_{\text{rt}}$, while the GTA step size (which is nearly equal to $h_{\text{rt}}$) has zero

[2] $x = 3.1$ has no exact finite binary representation.

error simply by fortuitous cancellation of roundoff errors. If the GTA step size is used with neighboring values of $x$, then its error should increase to the predicted value. In contrast, the AutoDX step size should still produce near-zero errors for neighboring $x$ values, since it was chosen from the middle of the zero-error step size region. These predictions are verified in Figure 4.2, in which it is seen that the GTA *eps* results in increased errors while the AutoDX *eps* retains error near machine precision.

**Example 4.2.** The function

$$f(x) = \frac{1}{3}x^3 - \frac{3}{2}x^2 + 2x + 1 \tag{4.3}$$

is differentiated at $x = 3.1$, using the $O(h^2)$ central-differences method. The initial large step size for the AutoDX algorithm is taken to be $h_0 = 1 + |x|$, and the rule-of-thumb step size is $h_{rt} = 5e{-}6|x|$. The solution using the three comparison algorithms is given in Table 4.2, in which the AutoDX results are given at the corrected optimal step size. The AutoDX algorithm determined that the maximum valid step size is $h_{\max} = 3$, and the condition error in $f(x)$ is $\epsilon \approx 7.6e{-}16$.

Table 4.2: Solution for $df/dx$ from example 4.2.

| | $h_{\text{opt}}$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | $1.55e{-}5$ | $1.99e{-}11$ | $1.04e{-}10$ | 2 |
| AutoDX | $1.53e{-}5$ | $2.42e{-}11$ | $1.01e{-}10$ | 73 |
| GTA | $7.79e{-}6$ | $4.69e{-}12$ | $2.62e{-}11$ | 164 |

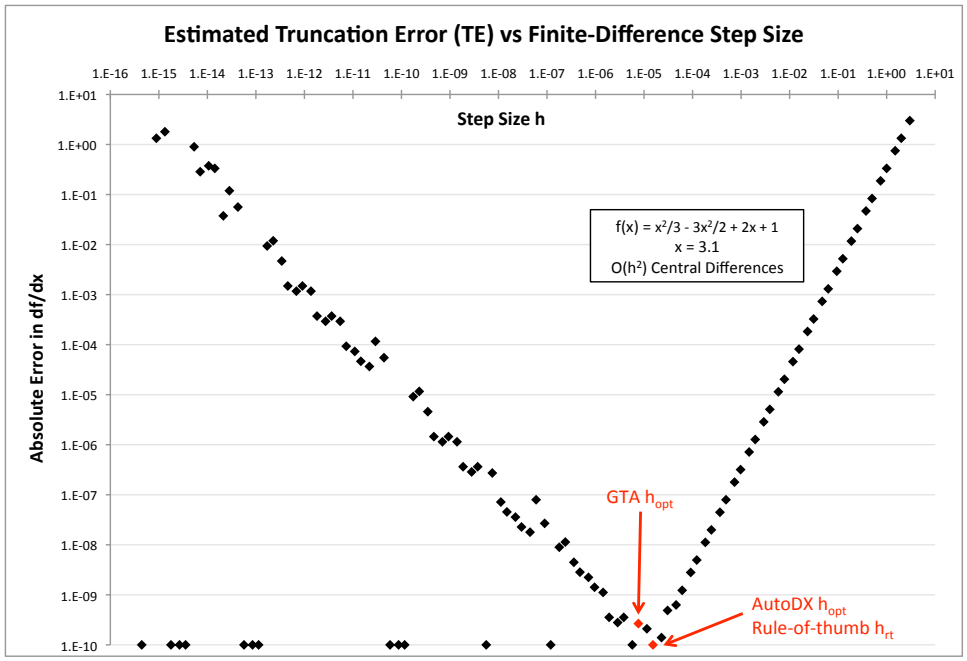


Figure 4.3: Estimated truncation errors for example 4.2.

These results are no surprise. The function is a simple polynomial with machine precision output evaluated at a near-unity $x$, so the rule-of-thumb step size is expected to be very close to the true optimal step size. Furthermore, in this case the FDD method's truncation error depends only on the constant third derivative, so the maximum valid step size should be equal to the largest tested step size. These predictions are confirmed by the truncation errors shown in Figure 4.3.

Since all three step sizes are close to each other, it is expected that the variation in errors with neighboring values of $x$ should be the same for each. Figure 4.4 confirms these predictions, which means that all three methods produce

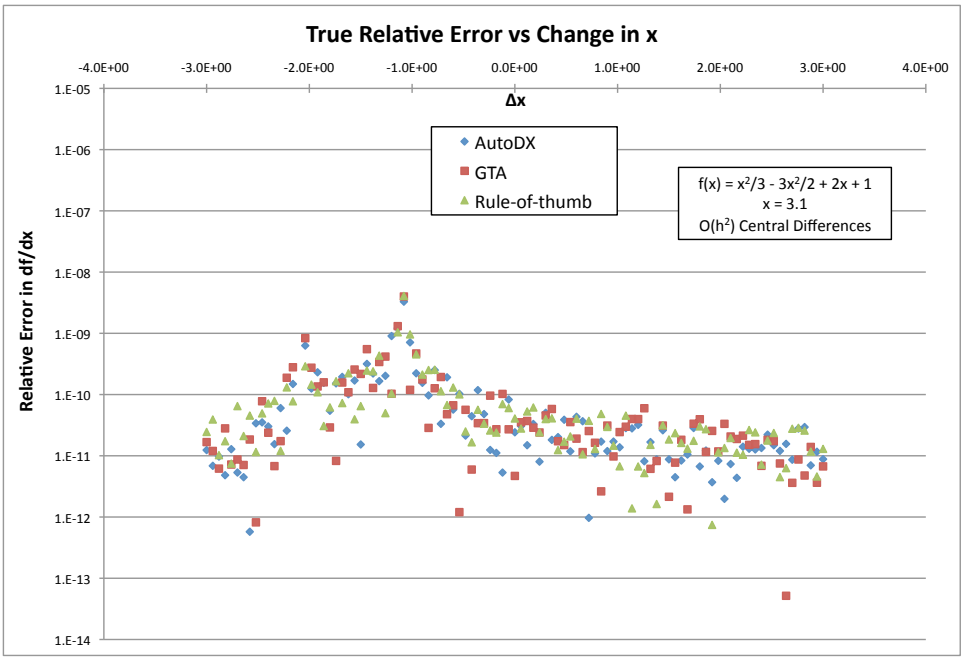


Figure 4.4: Relative errors with respect to neighboring $x$ values for example 4.2.

equivalent results for simple non-deviant polynomials. The only differences are the number of function evaluations. Of interest in this graph is the almost asymptotic increase in relative error near $\Delta x \approx -1.0$. The exact derivative of the function in this example is $f'(x) = (x-1)(x-2)$, which has a zero at $x = 2$. This corresponds to $\Delta x = -1.1$, which is exactly where the asymptotic increase in error occurs. For functions $f(x)$ evaluated at points $x$ at which the derivative is zero, it is common for the optimal step size to be different at neighboring $x$ values. In this case, AutoDX computes the optimal step size for the point $x = 2$ (i.e. $\Delta x = -1.1$) to be $h_{\text{opt}} = 8.73e{-}6$.

**Example 4.3.** The periodic function

$$f(x) = sin(x)cos(3x) \tag{4.4}$$

is differentiated at $x = -3.95$, using the $O(h^2)$ central-differences method. The initial large step size for the AutoDX algorithm is taken to be $h_0 = 1 + |x|$, and the rule-of-thumb step size is $h_{rt} = 5e{-}6|x|$. The solutions using the three comparison algorithms are given in Table 4.3, with the AutoDX results computed using the corrected optimal step size. The AutoDX algorithm determined that the maximum valid step size $h_{\max} = 0.25$, and the condition error in $f(x)$ is less than machine precision.

Table 4.3: Solution for $df/dx$ from example 4.3.

| | $h_{\text{opt}}$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | $1.98e{-}5$ | $1.06e{-}9$ | $3.19e{-}9$ | 2 |
| AutoDX | $1.91e{-}6$ | $1.26e{-}12$ | $2.96e{-}11$ | 85 |
| GTA | $4.95e{-}7$ | $2.06e{-}11$ | $1.04e{-}14$ | 164 |

The estimated truncation error plot for this problem, given in Figure 4.5, shows that the rule-of-thumb step size is clearly too large even though the function is well-conditioned. In this case, even if the alternate rule-of thumb $h_{rt} = 1e{-}7$ were to be used, the error would be well into the roundoff error range. One seemingly odd result in Figure 4.5 is that the GTA optimal step size seems to have almost zero estimated error, and Table 4.3 indicates an increased true relative error. This discrepancy is explained by the fact that the GTA algorithm honed in on a step-size region with very small dispersion due to coincidental roundoff error

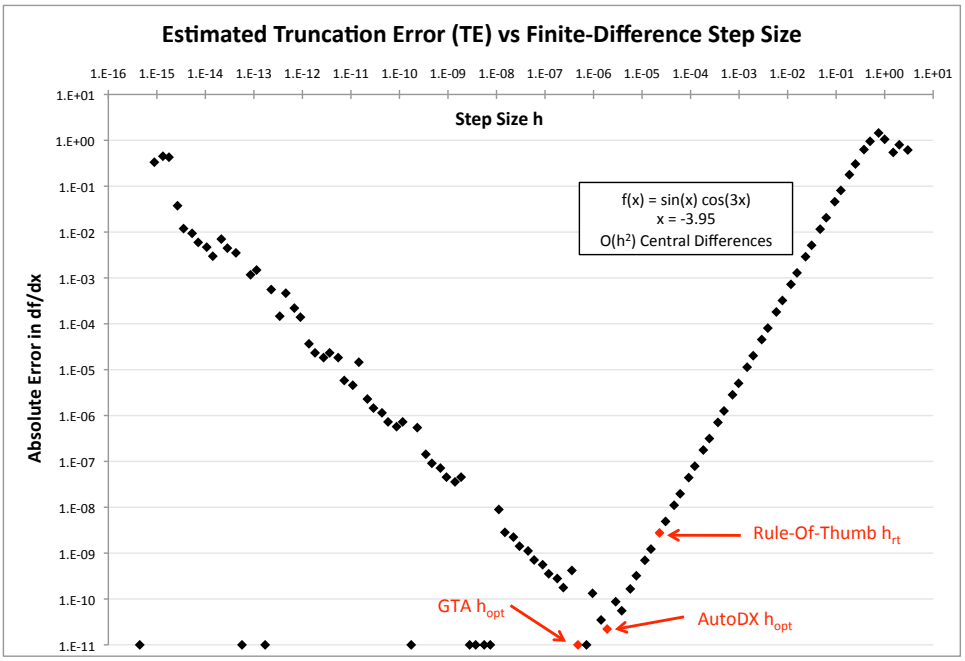


Figure 4.5: Estimated truncation errors for example 4.3.

cancellations. Such a repeated occurrence of nearly equal FDD values causes the $C_n$ estimate from (2.33) – and therefore also the truncation error estimate from (2.34) – to be nearly zero. This is a prime example of how roundoff errors can make the true error *appear* to be very small, and is precisely why step sizes smaller than the true $h_{\text{opt}}$ (as approximated by the corrected AutoDX $h_{\text{opt}}$) should be avoided.

The relative errors for neighboring values of $x$ are given in Figure 4.6. As explained in Section 3.5.2, the large rule-of-thumb step size $h_{\text{rt}}$ produces FDD values which are very precise but not very accurate. It can be seen that on average, the rule-of-thumb FDD errors are over an order of magnitude higher than the AutoDX step size errors. Since the GTA step size is just slightly smaller

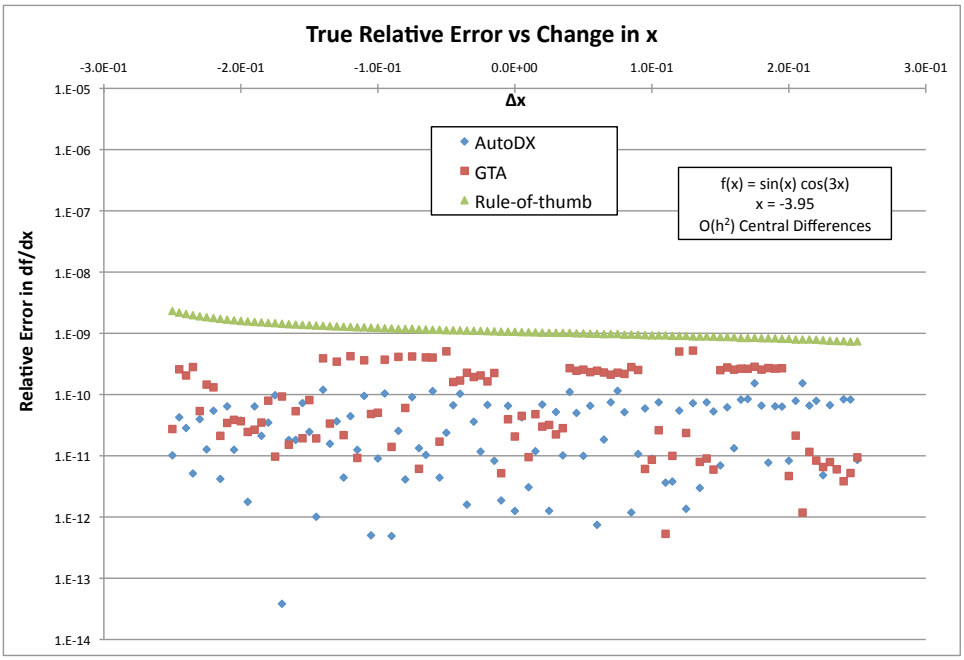


Figure 4.6: Relative errors with respect to neighboring $x$ values for example 4.3.

than the AutoDX step size, its errors are slightly higher, as expected.

**Example 4.4.** The highly nonlinear function

$$f(x) = \frac{e^x}{\sqrt{\sin x^3 + \cos x^3}} \tag{4.5}$$

has an asymptote at $x \approx 1.33067$. The first derivative is approximated at $x = 1.33$ using the $O(h^2)$ central-differences method, with an initial large step size for the AutoDX algorithm of $h_0 = 1 + |x|$. The solutions using the three comparison algorithms are given in Table 4.4. The AutoDX algorithm determined that the maximum valid step size $h_{\max} = 1.83e{-}4$, and the condition error in $f(x)$ is $\epsilon = 5.49e{-}14$.

Table 4.4: Solution for $df/dx$ from example 4.4.

| | $h_{\text{opt}}$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | $6.65e{-6}$ | $4.56e{-5}$ | $1.37e{-4}$ | 2 |
| AutoDX | $2.98e{-8}$ | $1.08e{-9}$ | $3.71e{-9}$ | 105 |
| GTA | $3.26e{-8}$ | $1.89e{-9}$ | $4.44e{-9}$ | 124 |

This example is interesting for two main reasons. First, the point $x$ is very close to an asymptote, so the maximum allowable step size $h_{\max}$ must be very small to not overstep the asymptote. As a result, AutoDX would have to be called within just a few iterations of an enclosing optimization loop as soon as $x'$ leaves the valid range given by $[x \pm h_{\max}]$. This proximity to an asymptote also causes the output of the function itself to lose a small amount of reliability. For this function, the $\epsilon \approx 10^{-13}$ implies that almost 3 digits of accuracy are lost in the computation of $f$ itself.

Secondly, step sizes just beyond the asymptote result in a complex-valued $f$. In programming practice, this has one of two results: either the system throws a runtime error, or the computation returns $\pm\infty$ or NaN (Not a Number). Throwing a runtime error is a specified behavior in all systems; there is generally a compiler flag or error handler that must first be set. The AutoDX algorithm has provisions to automatically disable this behavior during step-size analysis, so that all erroneous values are correctly indicated as $\pm\infty$ or NaN[3]. In addition, once a

[3]AutoDX cannot work properly if the implementation of $f(x)$ produces a fatal crash as a result of erroneous internal computations. Such functions are not considered here, since proper programming guidelines imply gracious error handling.

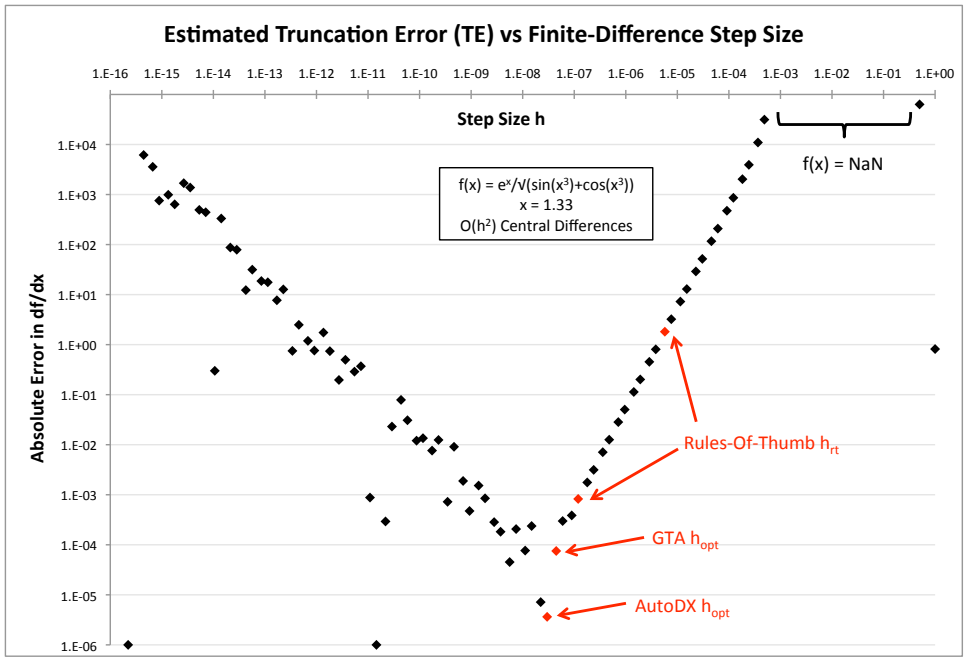


Figure 4.7: Estimated truncation errors for example 4.4.

non-numeric $f$ output is detected, AutoDX immediately skips to the next step size. By implementing these two simple techniques, AutoDX is able to graciously disregard step sizes which throw $f(x \pm h)$ out of the real domain.

The truncation error plot in Figure 4.7 shows the step-size region in which $f(x)$ has non-numeric output. AutoDX skips these step sizes entirely, without computing or analyzing estimated truncation errors for them. Of equal interest is the rule-of-thumb step size, which is well into the truncation error range for this function. If the alternate rule-of-thumb $h_{\text{rt}} = 1e{-}7|x| = 1.33e{-}7$ is chosen, then the estimated error would be much lower. However, this step size would not suitable for example 4.3 as seen in Figure 4.5. These two examples alone

highlight the fundamental problem with choosing a single step size for an unknown function within an optimization loop. The amount of trial-and-error required to find a 'good' step size can become staggering with more complicated functions evaluated over large ranges of $x$.

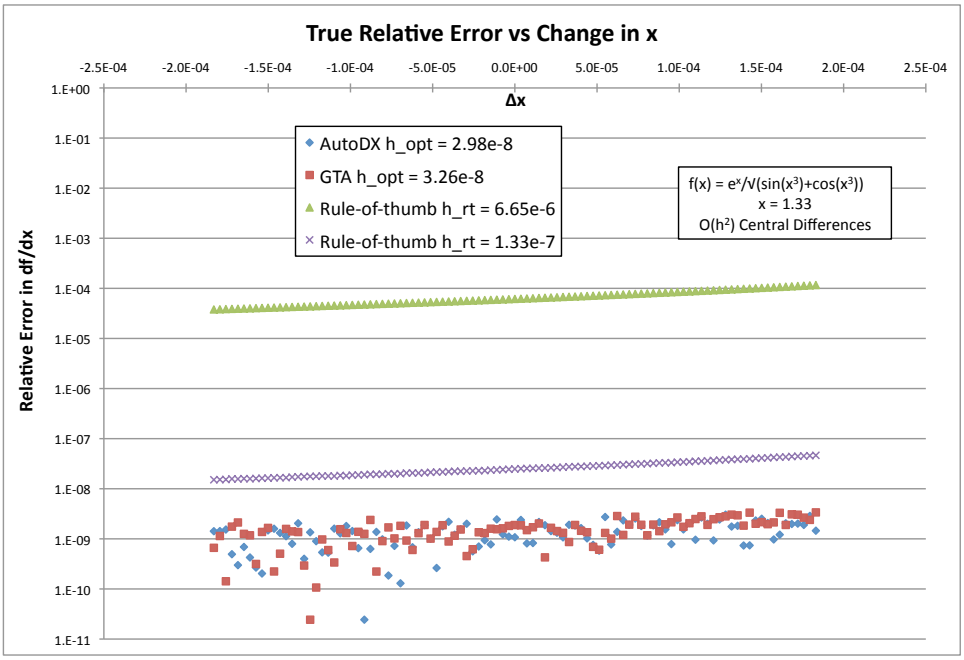


Figure 4.8: Relative errors with respect to neighboring $x$ values for example 4.4.

Figure 4.8 confirms the expected behavior for both rule-of-thumb step sizes. Although the smaller $h_{\text{rt}}$ value does produce considerably lower errors, those errors are still an order of magnitude greater than both AutoDX and GTA methods. As expected both of these latter two methods have equivalent errors for all $x$ values within the range given by $h_{\text{max}}$.

## 4.3 Examples of Algorithmic Functions

Algorithmic functions are those which require an implementation of a non-trivial algorithm to produce their results. Although not strictly required, algorithmic functions often employ iterative methods to produce a result. Because numerical errors can compound within these iterations, algorithmic functions are expected to have a greater condition error than the simpler fundamental functions of Section 4.2.

**Example 4.5.** The solution to Kepler's equation, briefly examined in Section 3.5.1, is now thoroughly considered using the various step-size estimation algorithms. The orbital parameters are given in Table 3.2, with the exception that the orbit is propagated to $x = t_f = 444067.6[s]$, which is 1000 seconds short of a half-period. The first derivative $d\nu_f/dt_f$ is approximated using the $O(h^4)$ central-differences method, with the initial large step size for the AutoDX algorithm taken as $h_0 = 1 + |x|$. Table 4.5 gives results from the three step-size approximation algorithms. AutoDX determined that $h_{\max} = 9830.4[s]$, and $\epsilon = 1.35e{-}13$.

Table 4.5: Solution for $d\nu_f/dt_f$ from example 4.5.

| | $h_{\text{opt}}[s]$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | 2.220 | $2.74e{-}7$ | $2.51e{-}7$ | 4 |
| AutoDX | 1024 | $5.75e{-}9$ | $1.62e{-}10$ | 57 |
| GTA | 488.5 | $1.64e{-}10$ | $1.74e{-}9$ | 368 |

The most notable result here is the large condition error $\epsilon$ in $f(x)$. To understand this, the implementation of $f(x)$ must be considered:

```
nu_f = solve_kepler(t_0, nu_0, t_f) {
  // Convert true to mean anomaly, accurate to machine precision
  M_f = convert_TA_to_MA(nu_0, t_f - t_0);

  // Compute final ecc anomaly, Danby's method accurate to 4e-16
  E_f = solve_kepler_danby(M_0, ecc);

  // Compute final true anomaly, using standard arccos method
  nu_f = acos((ecc - cos(E_f))/(ecc*cos(E_f) - 1));
  nu_f = correct_for_quadrant(nu_f, M_f); // No loss of precision
}
```

Danby's method of solving Kepler's equation is accurate almost to machine precision (see Section 3.5.1). By elimination, the culprit of the large condition error $\epsilon$ must be the `acos` function. Indeed, the implementation of the arccosine function is ill-conditioned at angles near 0 and $\pi$, the latter of which is the case in this example (since the final time is very close to the orbit's half-period).

To fix this problem, the quadrant-aware `atan2` function should be used in place of `acos`.

```
nu_f = solve_kepler(t_0, nu_0, t_f) {
  ...
  // Compute final true anomaly, using quadrant-aware arctangent method
  nu_f = atan2(sqrt(1.d0 - ecc^2)*sin(E_f), cos(E_f) - ecc);
```

```
}
```

The results from the three step-size approximation algorithms using this new method are given in Table 4.6. It is clear that the errors in $d\nu_f/dt_f$ are much lower, as expected with a FDD method of O($h^4$) accuracy. Furthermore, AutoDX now computes the condition error $\epsilon$ of $f(x)$ to be below machine precision, which verifies the increased accuracy of using `atan2` over `acos`.

Table 4.6: Solution for $d\nu_f/dt_f$ (using atan2) from example 4.5.

| | $h_{\text{opt}}[s]$ | \|True Rel Err\| | \|Est Rel Err\| | Num f() |
|---|---|---|---|---|
| Rule-of-thumb | 2.220 | $5.10e{-12}$ | $1.12e{-10}$ | 4 |
| AutoDX | 256.0 | $6.05e{-13}$ | $6.31e{-13}$ | 73 |
| GTA | 710.5 | $6.58e{-12}$ | $1.15e{-11}$ | 406 |

Figures 4.9 and 4.10 verify the validity of the AutoDX $h_{\text{opt}}$ value. Not only does it properly find the minimum of the estimated truncation error plot, but the resulting true relative errors for neighboring values of $x$ ($t_f$) have similar accuracy. It is also clear that the rule-of-thumb step size $h_{\text{rt}}$ simply does not apply here; it is well within roundoff error range for all tested $x$ values. Finally, it should be noted that even the largest tested step size $h_0$ ($\approx t_f$) falls within the truncation error range in Figure 4.9. This simply means that the maximum reliable step size $h_{\text{max}}$ is most likely larger than $h_0$. If the true $h_{\text{max}}$ is desired, then $h_0$ should be increased when running AutoDX.

This example shows the importance of being able to accurately estimate a function's condition error $\epsilon$. Engineering programmers sometimes use algorithms

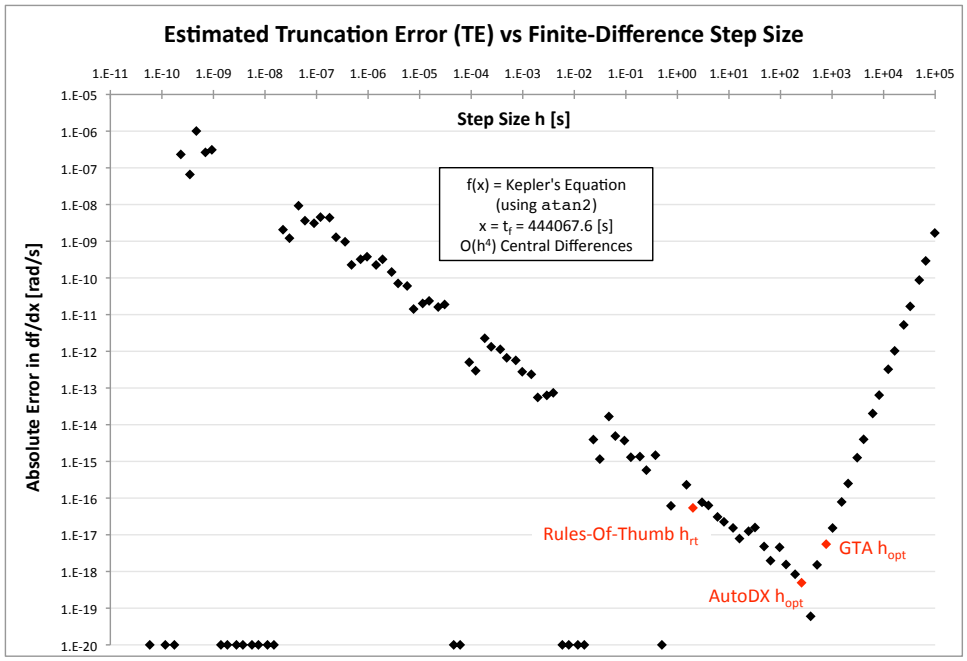


Figure 4.9: Estimated truncation errors for example 4.5 (using atan2).

given by reputable sources[4], without full consideration of the limits of these algorithms. An estimate of $\epsilon$ aids in verifying or challenging the fitness of a particular function implementation.

**Example 4.6.** To showcase the performance of AutoDX for a multidimensional step-size optimization problem, a Lunar intercept problem is considered. Given an initial Earth-centered orbit ($\mu = 398600.4[km^3/s^2]$), an optimal 2-body transfer to the Moon is computed. The initial and Lunar orbits are frozen, and the epoch time $t_0$, initial Lunar true anomaly $\nu_{0,Moon}$, and initial orbit true anomaly $\nu_0$ are all specified. The satellite coasts on the initial orbit for time $dt_1$, after which an

[4]In this case, the acos method comes from the Bate, Mueller, and White [2] book itself.

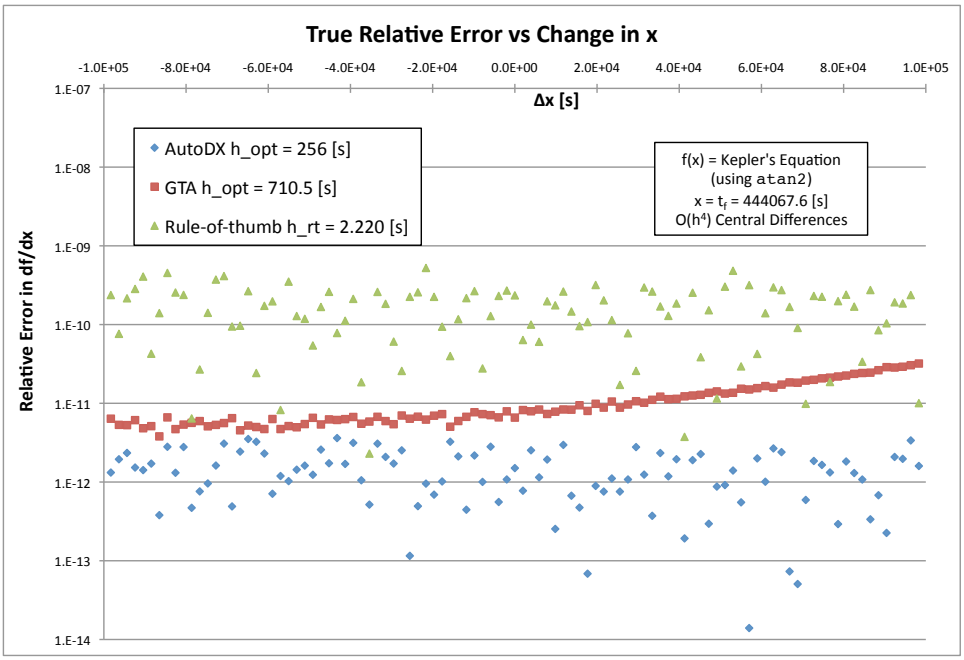


Figure 4.10: Relative errors with respect to neighboring $x$ values for example 4.5 (using atan2).

impulse $\Delta\mathbf{v}$ is applied to transfer the satellite to a transfer orbit. After coasting on the transfer orbit for time $dt_{ga}$, the satellite arrives at the Moon at $t_f = t_0 + dt_1 + dt_{ga}$. This setup is illustrated in Figure 4.11. The performance index for this problem is

$$J = \|\Delta\mathbf{v}\| = \Delta v \tag{4.6}$$

and the optimization variable and constraint vectors are

$$\mathbf{x}^\top = \begin{pmatrix} dt_1 & dt_{ga} & \Delta\mathbf{v}_x & \Delta\mathbf{v}_y & \Delta\mathbf{v}_z \end{pmatrix}_{1\times 5} \tag{4.7}$$

$$\mathbf{c} = \begin{pmatrix} \mathbf{r}_{\mathbf{Moon}}(t_f) - \mathbf{r}(t_f) = \mathbf{0} \end{pmatrix}_{3\times 1} \tag{4.8}$$

Table 4.7: Initial and Lunar orbits for example 4.6.

| | Initial Orbit | Lunar Orbit |
|---|---|---|
| $a$ | $24555.6[km]$ | $384400.0[km]$ |
| $e$ | 0.731921 | 0.0549 |
| $i$ | $51.619°$ | $19.0°$ |
| $\omega$ | $45.0°$ | $0.0°$ |
| $\Omega$ | $250.0°$ | $0.0°$ |
| $\nu_0$ | $0.0°$ | $90.0°$ |

While the system as stated is autonomous, the fact that time implicitly plays a role in the relative alignments of the satellite and Moon makes the solution nontrivial. An initial guess can be determined by assuming a Hohmann-like transfer orbit, with the transfer orbit plane coincident with the initial orbit plane[5]. While this transfer orbit does intercept the Moon's orbit, the Moon may not be at this location due to timing issues. This position error $\mathbf{c}$ can be minimized by considering orbit synchronization, but it will always be nonzero at the final time $t_f$ (with rare exceptions of aligned initial and Lunar orbits).

The optimization process drives $\mathbf{c}$ to zero while minimizing the total cost ($\Delta v$), and requires the gradients of both of these quantities with respect to the optimization variables $\mathbf{x}$. The latter gradient is easily computed analytically,

$$\frac{\partial J}{\partial \mathbf{x}} = \begin{pmatrix} 0 & 0 & \frac{\Delta \mathbf{v}}{\Delta v}^\top \end{pmatrix}_{1\times 5} \tag{4.9}$$

The Jacobian of the constraint function $\dfrac{\partial \mathbf{c}}{\partial \mathbf{x}}$ is much more difficult to compute.

[5]Since this is an intercept problem, it is assumed that there is no plane change at departure. The optimization process adds a small plane change when necessary.

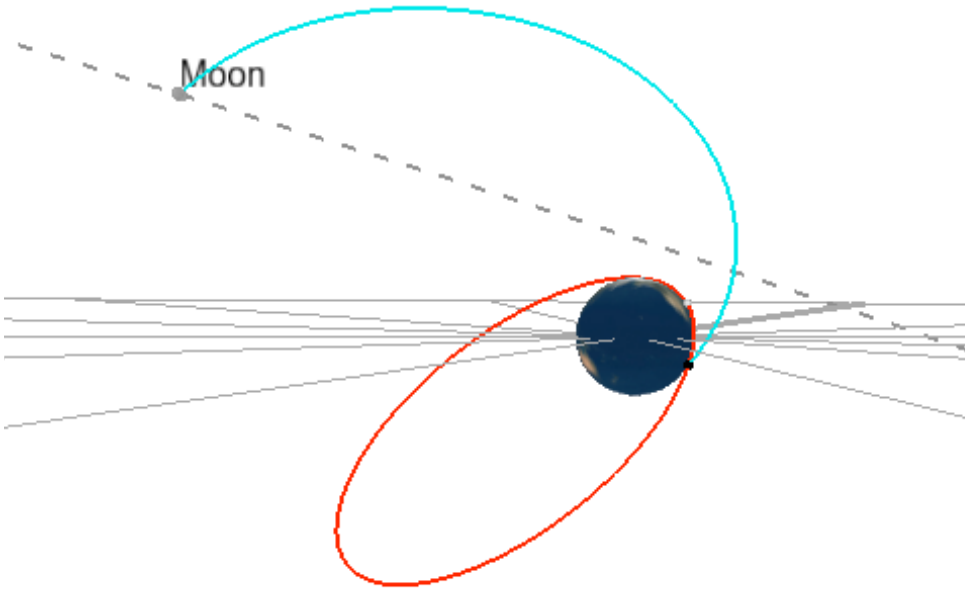


Figure 4.11: Setup of the Lunar transfer problem from example 4.6. The initial orbit is in red, and the transfer orbit is in blue.

Because this problem involves purely ballistic trajectory arcs, the state transition matrix approach of Goodyear [16, 17] or the variational method of Ocampo and Munoz [35] can be used to compute a near-analytical Jacobian matrix. However, it is much faster in terms of programming complexity and runtime to use a FDD method.

Continuing in the vein of previous examples, an $O(h^2)$ central-difference FDD method is used to compute each element of the Jacobian matrix $\frac{\partial \mathbf{c}_j}{\partial \mathbf{x}_i}$ ($1 \leq i \leq 5$, $1 \leq j \leq 3$). The reference $\mathbf{x}$ is the Hohmann transfer initial guess,

$$\mathbf{x}_0^\top \approx \begin{pmatrix} 37391 & 414135 & -1.34 & -4.09 & 0.18 \end{pmatrix}_{1\times 5} \tag{4.10}$$

where times are in $[s]$ and velocities are in $[km/s]$. The rule-of-thumb step size

for each element of $\mathbf{x}$ is $h_{i,\text{rt}} = 10^{-6}(1 + |\mathbf{x}_i|)$. For AutoDX, the initial large step sizes are $h_{i,0} = 1 + |\mathbf{x}_i|$. Since AutoDX computes the full gradient vector for a particular element $i$ of $\mathbf{x}$, it must be run in a loop over each $i$. On the other hand, GTA computes an individual partial derivative (for a given $i$ and $j$), so it must be run in a double nested loop over each $i$ and $j$. As discussed in Section 3.7, there may be a separate optimal step size $h_{\text{opt}}$ associated with each partial derivative $\frac{\partial \mathbf{c}_j}{\partial \mathbf{x}_i}$. Table 4.8 gives each of these step sizes as computed by AutoDX and GTA, as well as the single rule-of-thumb step size for each $\mathbf{x}_i$.

Table 4.8: Optimal step sizes for each element of $\mathbf{x}$ from example 4.6, given in $[s]$ for $dt$ and $[km/s]$ for $\Delta v$.

| | | $\mathbf{c}_1$ | $\mathbf{c}_2$ | $\mathbf{c}_3$ |
|---|---|---|---|---|
| | Rule-of-Thumb $h_{\text{rt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ |
| $dt_1$ | $3.74e{-2}$ | $4.88e{-4}$<br>$5.23e{-4}$ | $2.44e{-4}$<br>$4.11e{-4}$ | $9.77e{-4}$<br>$4.86e{-3}$ |
| $dt_{ga}$ | $4.14e{-1}$ | 2.00<br>6.21 | 1.00<br>7.45 | 2.00<br>6.63 |
| $\Delta\mathbf{v}_x$ | $2.34e{-6}$ | $3.81e{-6}$<br>$4.21e{-6}$ | $1.91e{-6}$<br>$3.04e{-6}$ | $1.91e{-6}$<br>$4.21e{-6}$ |
| $\Delta\mathbf{v}_y$ | $5.09e{-6}$ | $9.54e{-7}$<br>$7.13e{-6}$ | $9.54e{-7}$<br>$8.15e{-6}$ | $7.63e{-6}$<br>$7.64e{-5}$ |
| $\Delta\mathbf{v}_z$ | $1.18e{-6}$ | $1.91e{-6}$<br>$1.18e{-6}$ | $2.38e{-7}$<br>$1.30e{-6}$ | $1.91e{-6}$<br>$1.18e{-6}$ |

Note that while the gradient $\dfrac{\partial \mathbf{c}}{\partial dt_{ga}}$ can be computed analytically,

$$\frac{\partial \mathbf{c}}{\partial dt_{ga}} = \frac{\partial \mathbf{r}_{\text{Moon}}(t_f)}{\partial dt_{ga}} - \frac{\partial \mathbf{r}(t_f)}{\partial dt_{ga}} = \mathbf{v}_{\text{Moon}}(t_f) - \mathbf{v}(t_f) \tag{4.11}$$

it is assumed here that none of the Jacobian elements are actually known. The

three step-size estimation methods are compared using the estimated truncation error plot alone, which (as proven in Chapter 2) is a good approximation of the true error. It can be seen from Table 4.8 that the variation in $h_{\text{opt}}$ between methods is greatest when $\mathbf{x}_4 = \Delta\mathbf{v}_y$ is used to compute the derivative of $\mathbf{c}_3 = \mathbf{r}_{\text{Moon},z}(t_f) - \mathbf{r}_z(t_f)$.

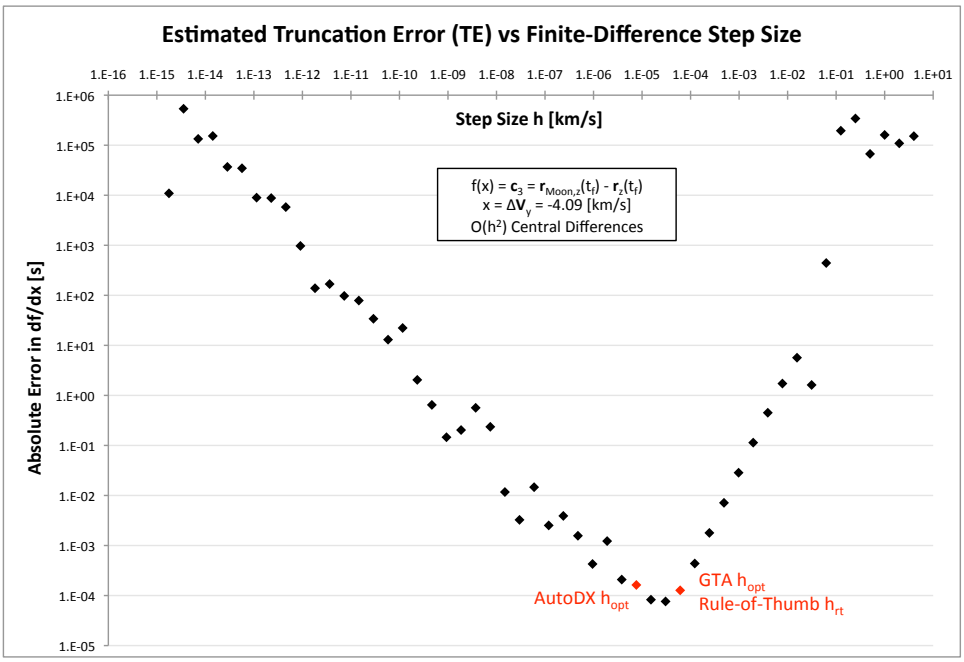


Figure 4.12: Derivative $\dfrac{\partial \mathbf{c}_3}{\partial \Delta \mathbf{v}_y}$ from example 4.6.

Figure 4.12 shows these optimal step sizes overlaid on the estimated truncation error plot for this particular derivative. Here, it seems as if the AutoDX $h_{\text{opt}}$ is too small, i.e. within roundoff error. However, AutoDX reports the corrected $h_{\text{opt}}$ using (2.83), which is expected to be smaller than the uncorrected $h_{\text{opt}}$ as proven in Section 2.9. The uncorrected step size for this case is $h_{\text{opt}} = 1.53e{-}5[km/s]$

which is almost exactly at the minimum of the (uncorrected) estimated truncation error plot.

Table 4.9: Condition errors $\epsilon_{ij}$ for each $\mathbf{c}_j$ with respect to each element of the input $\mathbf{x}_i$, from example 4.6.

| | $\mathbf{c}_1$ | $\mathbf{c}_2$ | $\mathbf{c}_3$ |
|---|---|---|---|
| $dt_1$ | $2.17e{-}14$ | $1.04e{-}14$ | $1.00e{-}13$ |
| $dt_{ga}$ | $3.83e{-}15$ | $1.73e{-}16$ | $6.45e{-}15$ |
| $\Delta\mathbf{v}_x$ | $6.93e{-}14$ | $3.12e{-}14$ | $6.94e{-}15$ |
| $\Delta\mathbf{v}_y$ | $1.81e{-}15$ | $8.05e{-}15$ | $2.71e{-}14$ |
| $\Delta\mathbf{v}_z$ | $4.54e{-}14$ | $2.27e{-}16$ | $3.13e{-}14$ |

The condition errors as computed by AutoDX are given in Table 4.9. For multivariate functions such as $\mathbf{c}$, which depend on multiple inputs, the condition error $\epsilon_{ij}$ for any given $\mathbf{c}_j$ will depend on which input $\mathbf{x}_i$ is being varied. It can be seen that the largest condition error occurs for the function $\mathbf{c}_3$ with respect to $dt_1$. $dt_1$ determines the location of the trans-Lunar $\Delta\mathbf{v}$ impulse, which is defined in the inertial reference frame. It is therefore expected that $\mathbf{c}_3$ will be mathematically sensitive to changes in $dt_1$; in fact, the derivative $\dfrac{\partial \mathbf{c}_3}{\partial dt_1}$ is over $4100[km/s]$. More importantly, however, is the fact that the actual computation of $\mathbf{c}$ with respect to a change in $dt_1$ involves multiple Kepler propagations and coordinate transformations, which accumulate error. The combination of mathematical sensitivity and algorithmic error accumulation leads to this increased condition error. In situations like this, a good approach to reducing condition error would be to redefine $\Delta\mathbf{v}$ in a coordinate system relative to the local velocity vector. In this case, small errors in the location of the $\Delta\mathbf{v}$ (due to the propagation from $t_0$ to $t_0 + dt_1$) would

have a smaller effect on the final position error $\mathbf{c}$.

Table 4.10: Number of function evaluations in computing $\dfrac{\partial \mathbf{c}}{\partial \mathbf{x}_i}$, from example 4.6.

| | Rule-of-Thumb | AutoDX | GTA |
|---|---|---|---|
| $dt_1$ | 2 | 57 | 392 |
| $dt_{ga}$ | 2 | 39 | 532 |
| $\Delta\mathbf{v}_x$ | 2 | 43 | 492 |
| $\Delta\mathbf{v}_y$ | 2 | 47 | 492 |
| $\Delta\mathbf{v}_z$ | 2 | 47 | 512 |
| Total | 10 | 243 | 2420 |

The number of function evaluations for each method is given in Table 4.10. As expected, using a fixed step size completely outperforms all other methods. GTA has a high number of function evaluations for two reasons: it must be run in a doubly-nested loop, and many function evaluations must be performed to get good statistical approximations. The computational cost of AutoDX shown here is a worst-case scenario where the initial step size $h_0$ is taken to be very large. In practical use, such a large step size would only be used once; it could be reduced to the maximum valid step size $h_{\max}$ for future invocations of AutoDX. For this example, the $h_{\max}$ for each $\mathbf{x}_i$ is given in Table 4.11. If the initial step size is taken to be slightly larger than this maximum, e.g. $h_{0,i} = 10h_{\max,i}$, then the total number of function evaluations for AutoDX reduces to 203. In practice, it has been observed that the cost savings of using $h_0 = 10h_{\max}$ (for later invocations of AutoDX) increases as the problem complexity increases, mostly because the maximum valid step-size reduces as the problem becomes more nonlinear and the

function accumulates more error.

Table 4.11: Maximum valid step size $h_{\max}$ for each element of $\mathbf{x}$, from example 4.6. Given in $[s]$ for $dt$, and $[km/s]$ for $\Delta v$.

| | $dt_1$ | $dt_{ga}$ | $\Delta\mathbf{v}_x$ | $\Delta\mathbf{v}_y$ | $\Delta\mathbf{v}_z$ |
|---|---|---|---|---|---|
| $h_{\max}$ | 8 | 262144 | $6.25e{-2}$ | $7.81e{-3}$ | $3.13e{-2}$ |

It should be noted that the $h_{\max}$ value for $dt_1$ is quite small (8 seconds) as compared to the actual value of $dt_1$ (37391 seconds). Since the final position error $\mathbf{c}$ is very sensitive to the location of $\Delta\mathbf{v}$, the valid step-size region is small and $h_{\max}$ occurs closer to $h_{\text{opt}}$. As a result, AutoDX would have to be run more often for the $dt_1$ variable since the optimizer would most likely change it by more than 8 seconds fairly quickly.

**Example 4.7.** The three-finite-burn transfer problem previously studied by the author [33] is now analyzed to show the use of step size optimization for finite-burn problems. This problem involves using three maneuvers to transfer a satellite from an initial Lunar orbit to an escape $\mathbf{v}_\infty$ vector, which puts the satellite on an earthbound trajectory. This problem has been extensively considered in academia and for specific NASA missions, using both impulsive and finite-burn maneuvers [3, 8, 10, 14, 34–36, 45, 56, 57]. The particular variation of the problem used in this example is reproduced from the author's previous paper [33], and is analyzed here from the perspective of step-size optimization.

Figure 4.13 illustrates the geometry of the three-burn problem using finite-burn maneuvers. Starting in a Lunar orbit, the transfer sequence to a $\mathbf{v}_\infty$ vector

is divided into six segments. The spacecraft remains in the Lunar parking orbit from the initial epoch $t_0$ to the start of the first burn $t_{b10}$. The first burn continues until $t_{b11}$, after which the spacecraft coasts until the start of the second burn at $t_{b20}$. The second burn continues until $t_{b21}$, after which the spacecraft again coasts until the start of the third burn at $t_{b30}$. At $t_{b31}$, the third burn finishes and the spacecraft is on the desired hyperbolic escape trajectory with hyperbolic excess velocity vector $v_\infty^*$. For each burn $i$, the finite-burn maneuver is taken to be inertially fixed over the duration $(t_{bi0}, t_{bi1})$, as shown in Figure 4.14.

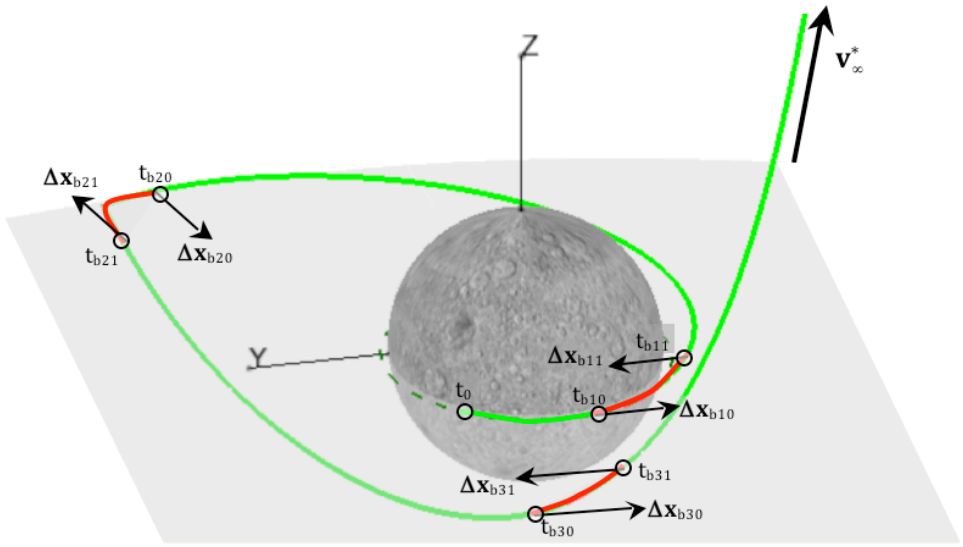


Figure 4.13: The three-finite-burn transfer from an initial orbit (at $t_0$) to a $\mathbf{v}_\infty$ vector, from example 4.7.

The objective function for this problem is

$$J = \text{minimize}(-m_f) \tag{4.12}$$

and the optimization variable and constraint function vectors are

$$\mathbf{x}^\top = \begin{pmatrix} t_{b10} & t_{b11} & t_{b20} & t_{b21} & t_{b30} & t_{b31} & \mathbf{u}_1^\top & \mathbf{u}_2^\top & \mathbf{u}_3^\top \end{pmatrix}_{1\times 15} \tag{4.13}$$

$$\mathbf{c} = \begin{pmatrix} \mathbf{v}_\infty(t_{b31}) - \mathbf{v}_\infty^* = \mathbf{0} \\ \|\mathbf{u}_1\| - 1 = 0 \\ \|\mathbf{u}_2\| - 1 = 0 \\ \|\mathbf{u}_3\| - 1 = 0 \\ t_{b11} - t_{b10} \geq 0 \\ t_{b21} - t_{b20} \geq 0 \\ t_{b31} - t_{b30} \geq 0 \\ r_p(t_{b21}) - r_{p,\min} \geq 0 \end{pmatrix}_{10\times 1} \tag{4.14}$$

Here, the final inequality constraint on the perilune distance $r_p$ is necessary in some cases to ensure that the transfer does not impact the Moon.

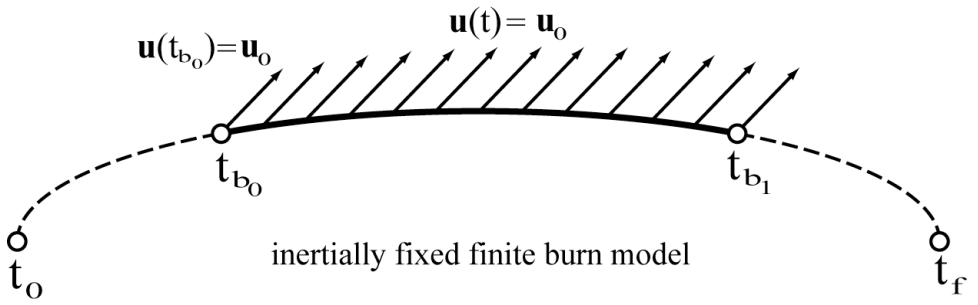


Figure 4.14: Parameters for the inertially-fixed finite-burn model from example 4.7.

A good initial guess solution for the finite-burn problem is obtained by first optimizing the impulsive-burn problem, the details of which are given by Ocampo and Munoz [35]. Each of the resulting optimal impulses are converted to finite burns via the rocket equation, which then form the initial guess $\mathbf{x}_0$. The resulting finite-burn trajectory no longer satisfies the constraints, so it is re-optimized using the above optimization variables and constraints. As shown in [33, 34], the

gradient of the objective function $\frac{dJ}{d\mathbf{x}}$ can easily be computed analytically. The gradients of the constraints $\frac{d\mathbf{c}}{d\mathbf{x}}$ are computed using an $O(h^2)$ central-difference FDD method, and evaluated at the initial guess $\mathbf{x}_0$ before finite-burn optimization is performed. As in all examples, the FDD step sizes are computed using a rule-of-thumb method, AutoDX, and GTA. Since the Variational Model from previous papers produces near-analytical gradients, it is used as the 'true' gradient for all comparisons.

The initial Lunar orbit is circular and equatorial with a radius of $1838[km]$ ($\approx 100[km]$ altitude). The target $\mathbf{v}_\infty^*$ vector has a magnitude of $1.2[km/s]$, with zero right ascension and 50° declination. The method of Jones and Ocampo [21] is used to generate an initial guess for the impulsive-burn optimization. This requires estimates of the post-TEI1 apolune and post-TEI2 perilune distances, which are specified as $r_{a,1} = 17000[km]$ and $r_{p,2} = 1839[km]$. The optimized impulsive trajectory is converted to a finite-burn trajectory using thrust magnitude $T = 32.5[kN]$ and specific impulse $I_{sp} = 320[s]$. The resulting initial guess optimization variable vector is

$$\mathbf{x}_0^\top = \begin{pmatrix} 4294 & 4601 & 59161 & 59267 & 88460 & 88636 & \mathbf{u}_1^\top & \mathbf{u}_2^\top & \mathbf{u}_3^\top \end{pmatrix} \tag{4.15}$$

where the times are in seconds. The inertially-fixed thrust direction unit vectors $\mathbf{u}_i$ are

$$\mathbf{u}_1^\top = \begin{pmatrix} 0.715 & -0.690 & 0.114 \end{pmatrix} \tag{4.16}$$

$$\mathbf{u}_2^\top = \begin{pmatrix} 0.262 & -0.436 & -0.861 \end{pmatrix} \tag{4.17}$$

$$\mathbf{u}_3^\top = \begin{pmatrix} 0.0856 & -0.657 & 0.749 \end{pmatrix} \tag{4.18}$$

When computing the gradients $\frac{d\mathbf{c}}{d\mathbf{x}}$, only the first three rows corresponding to $\frac{\partial \mathbf{v}_\infty(t_{b31})}{\partial \mathbf{x}}$ are of interest. It is shown in [33] that the remaining rows have trivial analytical expressions. Therefore, in this example only the gradients of the $\mathbf{v}_\infty$ constraints ($\mathbf{c}_{1-3}$) are considered. Furthermore, it is expected that parameters associated with the first finite-burn arc ($t_{b10}$, $t_{b11}$, $\mathbf{u}_1$) will have the largest effect on the $\mathbf{v}_\infty$ constraint since they occur earliest in the trajectory. Therefore, for this example only the $\mathbf{x}_1$ ($t_{b10}$), $\mathbf{x}_2$ ($t_{b11}$), and $\mathbf{x}_{7-9}$ ($\mathbf{u}_1$) optimization variables are considered.

Table 4.12: Optimal step sizes for the interesting elements of $\mathbf{x}$ and $\mathbf{c}$ from example 4.7, given in [$s$] for times and as nondimensional for thrust direction vectors.

| | | $\mathbf{c}_1$ | $\mathbf{c}_2$ | $\mathbf{c}_3$ |
|---|---|---|---|---|
| | Rule-of-Thumb $h_{\text{rt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ | AutoDX $h_{\text{opt}}$<br>GTA $h_{\text{opt}}$ |
| $t_{b10}$ | $1.00e{-4}$ | $3.84e{-5}$<br>$5.15e{-5}$ | $3.76e{-5}$<br>$4.73e{-5}$ | $3.78e{-5}$<br>$4.73e{-5}$ |
| $t_{b11}$ | $1.00e{-4}$ | $3.84e{-5}$<br>$5.98e{-5}$ | $2.39e{-5}$<br>$5.98e{-5}$ | $2.37e{-5}$<br>$8.28e{-5}$ |
| $\mathbf{u}_{1,x}$ | $1.00e{-6}$ | $6.01e{-7}$<br>$3.09e{-7}$ | $2.24e{-7}$<br>$3.09e{-7}$ | $3.73e{-7}$<br>$3.09e{-7}$ |
| $\mathbf{u}_{1,y}$ | $1.00e{-6}$ | $6.01e{-7}$<br>$2.53e{-7}$ | $2.24e{-7}$<br>$2.53e{-7}$ | $3.73e{-7}$<br>$3.04e{-7}$ |
| $\mathbf{u}_{1,z}$ | $1.00e{-6}$ | $1.92e{-5}$<br>$2.12e{-5}$ | $1.19e{-5}$<br>$1.45e{-5}$ | $7.18e{-6}$<br>$1.23e{-5}$ |

Table 4.12 gives the optimal step size for each optimization variable of interest, using the three step-size estimation methods. It should be noted that during the study done in [33], a considerable amount of time and effort was spent

in determining the rule-of-thumb step sizes. The results in Table 4.12 confirm that the $h_{rt}$ values are within an order of magnitude of the true optimal step sizes (from either AutoDX or GTA). In other words, AutoDX is able to determine optimal step sizes in a fraction of the time – and with significantly greater certainty – than the manual guess-and-check approach used traditionally.

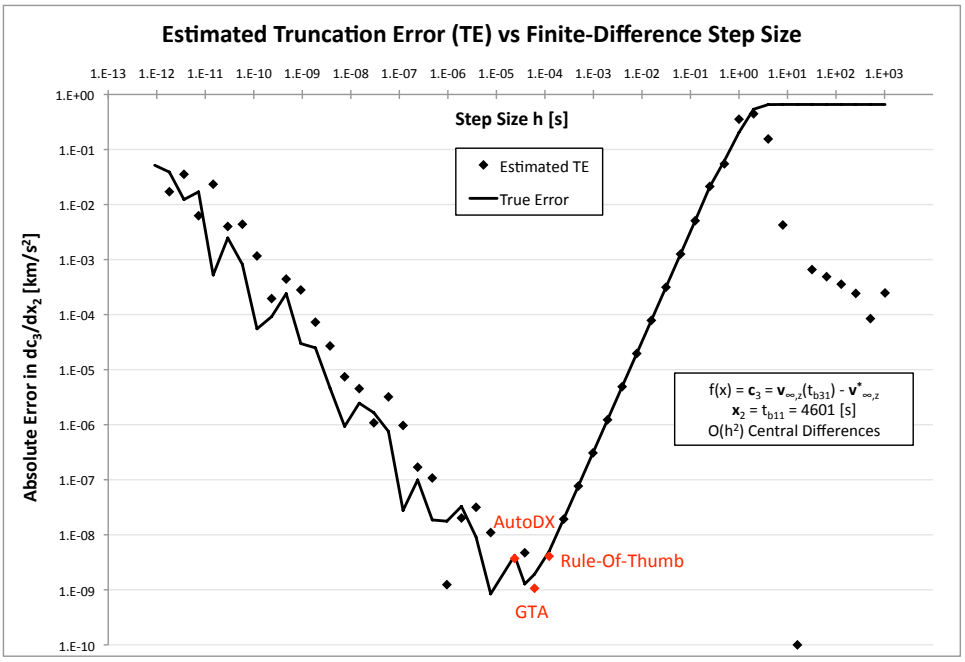


Figure 4.15: Estimated truncation errors for derivative $\dfrac{d\mathbf{c}_3}{d\mathbf{x}_2}$, from example 4.7.

For most of the optimization variables of interest, the step sizes computed by AutoDX and GTA are very similar. The most notable exception is in $t_{b11}$ (the end time of the first finite-burn arc), for which the AutoDX step size, corrected as per (2.83), is less than a third of the GTA step size. This discrepancy warrants a closer look at the estimated truncation errors and neighboring $\mathbf{x}_2$ behavior, given

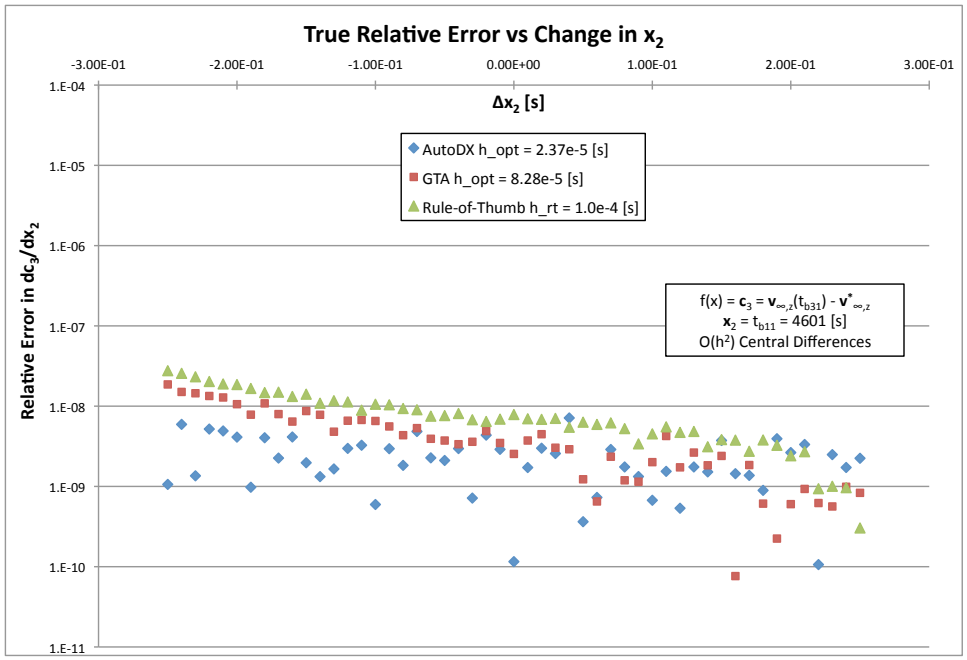


Figure 4.16: Relative errors for neighboring values of $\mathbf{x}_2$, from example 4.7.

in Figures 4.15 and 4.16. The truncation error plot seems to indicate that both GTA or AutoDX step sizes are equally close to the true minimum when only considering estimated truncation error. However, when this error is corrected using (2.78), it more closely matches the true error, and the true optimal step size is seen to be at (or very close to) the AutoDX estimate. Figure 4.16 also shows that the AutoDX step size produces consistently small relative errors when used with neighboring values of $\mathbf{x}_2$ ($t_{b11}$). In comparison, the GTA and rule-of-thumb step sizes do not provide this consistency, mainly because their larger-than-optimal values (albeit very slightly so) are more affected by mathematical truncation error.

The number of function evaluations performed by each step-size estimation

method is given in Table 4.13. As with example 4.6, the number of function evaluations for AutoDX can be reduced by using an initial step size $h_0$ slightly larger than the maximum safe step size. Table 4.14 shows that these maximum safe step sizes are very small; for the thrust direction unit vectors, they are only 3 orders of magnitude greater than the optimal step sizes. This means that when the optimal step size is used within an optimization loop, the AutoDX algorithm would have to be re-run fairly quickly as the optimizer changes a given parameter $\mathbf{x}_i$.

Table 4.15 indicates fairly high condition errors for the three constraint equality functions relative to the optimization parameters of the first finite-burn arc. In some cases, it is seen that almost four full digits of precision are lost in the computation of the constraint. However, unlike the Kepler problem of example 4.5, this loss of precision is not caused by programming errors. Rather, it is simply a consequence of the fact that the process of computing the final $\mathbf{v}_\infty$ is very sensitive to the first finite-burn arc parameters. Without a reformulation of the underlying mathematical equations of the problem (e.g. nondimensionalization), this high condition error cannot be avoided.

## 4.4 Effects on Numerical Optimization

The effectiveness of using an optimal step size has so far only been considered in terms of the accuracy of the derivative itself. However, it is rare for the derivative to be sought for its own sake; it is often used as part of a larger gradient-based optimization algorithm. When comparing various derivative-estimation

Table 4.13: Number of function evaluations in computing $\frac{\partial \mathbf{c}_{1-3}}{\partial \mathbf{x}_i}$ ($i \in \{1, 2, 7, 8, 9\}$), from example 4.7.

| | Rule-of-Thumb | AutoDX | GTA |
|---|---|---|---|
| $t_{b10}$ | 2 | 57 | 372 |
| $t_{b11}$ | 2 | 59 | 372 |
| $\mathbf{u}_x$ | 2 | 49 | 432 |
| $\mathbf{u}_y$ | 2 | 47 | 432 |
| $\mathbf{u}_z$ | 2 | 39 | 552 |
| Total | 10 | 251 | 2160 |

Table 4.14: Maximum valid step size $h_{\max}$ for each $\mathbf{x}_i$, from example 4.6. Given in $[s]$ for times and as nondimensional for thrust direction vectors.

| | $t_{b10}$ | $t_{b11}$ | $\mathbf{u}_x$ | $\mathbf{u}_y$ | $\mathbf{u}_z$ |
|---|---|---|---|---|---|
| $h_{\max}$ | 0.125 | 0.125 | $4.88e{-4}$ | $4.88e{-4}$ | $1.56e{-2}$ |

methods within an optimization problem, a performance metric must be chosen carefully. The arguments for (or against) several optimization metrics are now considered.

1. Number of optimization iterations. The accuracy of a derivative certainly affects the path taken by an optimization algorithm. If one derivative estimation method produces very inaccurate derivatives, then the optimizer may well take many more iterations to converge to a solution as compared to a derivative estimation method that produces accurate derivatives. On the other hand, given two derivatives with comparable accuracy (within a few orders of magnitude), there is no guarantee that the 'better' derivative

Table 4.15: Condition errors $\epsilon_{ij}$ for each $\mathbf{c}_j$ with respect to each element of the input $\mathbf{x}_i$, from example 4.7.

| | $\mathbf{c}_1$ | $\mathbf{c}_2$ | $\mathbf{c}_3$ |
|---|---|---|---|
| $t_{b10}$ | $2.97e{-}14$ | $1.96e{-}14$ | $3.09e{-}14$ |
| $t_{b11}$ | $2.95e{-}14$ | $2.34e{-}14$ | $3.73e{-}15$ |
| $\mathbf{u}_x$ | $9.61e{-}13$ | $1.01e{-}14$ | $1.25e{-}13$ |
| $\mathbf{u}_y$ | $8.31e{-}13$ | $6.92e{-}14$ | $1.08e{-}13$ |
| $\mathbf{u}_z$ | $2.11e{-}12$ | $1.49e{-}13$ | $7.40e{-}14$ |

will result in faster convergence than the 'worse' derivative. Because of this, using the number of optimization iterations to compare multiple derivatives (with comparable accuracy) is not always an accurate comparison.

2. Convergence accuracy. If the optimizer manages to reach the neighborhood of the minimum, then small variations in the accuracy of the derivatives do not affect the overall convergence accuracy of the solution. Therefore, convergence accuracy is not a good candidate for a metric with which to compare derivative-estimation methods.

3. Graph of step size versus optimization variable value. Rule-of-thumb step size estimation methods have a constant relationship between the variable and its step size. However, more sophisticated methods such as AutoDX and GTA assume no such relationship, and so it is of interest to observe how the various step-size estimation methods vary the step size as the optimization proceeds to convergence.

Of the three metrics discussed, only the step size vs optimization variable graph is used to compare the three step-size estimation methods in the context of an optimization loop. The other metrics are given only for completeness.

**Example 4.8.** A classic problem used to teach optimal control theory and trajectory optimization is the Lunar Lander problem. Given a spacecraft initially at rest at some point above the Lunar surface, a minimum-time trajectory is computed which results in a soft landing. The spacecraft's initial position is specified as $\mathbf{r}_0^\top = \begin{pmatrix} 1 & 1 \end{pmatrix}$ km, with a zero initial velocity $\mathbf{v}_0$. The desired final position is at the origin, with a zero final velocity. Spacecraft engine parameters are: $T_{\min} = 0$ N, $T_{\max} = 200$ N, $c = 10$ km/s. Lunar gravity is taken to be $g = 1.6$ m/s$^2$, and the initial spacecraft mass is $m_0 = 100$ kg. The differential equations describing system dynamics are,

$$\mathbf{x} = \begin{pmatrix} \mathbf{r} \\ \mathbf{v} \\ m \end{pmatrix} \qquad \dot{\mathbf{x}} = \begin{pmatrix} \mathbf{v} \\ \frac{T}{m}\hat{\mathbf{u}} - \begin{pmatrix} 0 \\ g \end{pmatrix} \\ \frac{-T}{c} \end{pmatrix} \qquad \hat{\mathbf{u}} = \begin{pmatrix} \hat{u}_x \\ \hat{u}_y \end{pmatrix} \ \|\hat{\mathbf{u}}\| = 1 \tag{4.19}$$

with initial conditions

$$\mathbf{x}(t_0) = \mathbf{x}_0 = \begin{pmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \\ m_0 \end{pmatrix} \tag{4.20}$$

Using the costates $\boldsymbol{\lambda}$, this optimization problem is transformed to a two-point boundary value targeting problem. The Hamiltonian function of the system is,

$$H = \boldsymbol{\lambda}_r^\top \mathbf{v} + \boldsymbol{\lambda}_v^\top \left( \frac{T}{m}\hat{\mathbf{u}} - \begin{pmatrix} 0 \\ g \end{pmatrix} \right) + \lambda_m \left( \frac{-T}{c} \right)$$

The optimization parameter vector, consisting of unknown initial point costates and the unknown final time, is

$$\mathbf{x}_p^\top = \begin{pmatrix} \lambda_{rx0} & \lambda_{ry0} & \lambda_{vx0} & \lambda_{vy0} & \lambda_{m0} & t_f \end{pmatrix}$$

subject to the costate differential equations

$$\dot{\boldsymbol{\lambda}}_r = 0 \tag{4.21}$$

$$\dot{\boldsymbol{\lambda}}_v = -\boldsymbol{\lambda}_r \tag{4.22}$$

$$\dot{\lambda}_m = \frac{-T}{m^2}\|\boldsymbol{\lambda}_v\| \tag{4.23}$$

Applying the Pontryagin Minimum Principle, the thrust direction is chosen to always be anti-parallel to the velocity costate $\boldsymbol{\lambda}_v$. The thrust magnitude is chosen according to the switching function,

$$S = -\left(\frac{\|\boldsymbol{\lambda}_v\|}{m} + \frac{\lambda_m}{c}\right) \tag{4.24}$$

$$T = \begin{cases} T_{\min} & S \geq 0 \\ \\ T_{\max} & S < 0 \end{cases} \tag{4.25}$$

The targeted final conditions are

$$\boldsymbol{\beta} = \begin{pmatrix} \mathbf{r}(t_f) \\ \mathbf{v}(t_f) \\ H(t_f) \\ \lambda_m(t_f) \end{pmatrix} = \mathbf{0} \tag{4.26}$$

This constitutes a system of 6 initial-point unknowns $\mathbf{x}_p$ and 6 final-point constraints $\boldsymbol{\beta}$. While the $\lambda_{m0}$ initial value and $\lambda_{mf}$ constraint equation can be eliminated, they are kept in this example for completeness.

In order to solve this targeting problem, an initial guess must be obtained for the unknown optimization parameters $\mathbf{x}_p$. Because the $\boldsymbol{\lambda}$ costates are unintuitive, the adjoint control transformation is often used to obtain initial values.

While this method does help in obtaining an initial guess for the optimizer, the quality of the computed derivatives are also important to the optimization process.

Derivatives for this example are computed using the rule-of-thumb method ($h_{rt} = 1e{-}6(1 + |x|)$), and using two variations of AutoDX. The first variation involves calling AutoDX at every optimization iteration to obtain $h_{\text{opt}}$. The second variation involves only calling AutoDX when the optimization variable changes by the maximum valid step size $h_{\max}$, as explained in Section 3.5.3. The first method ensures that the computed gradients are as accurate as possible at every optimization iteration, while the second method saves many function calls by re-using the step size until it must be changed.

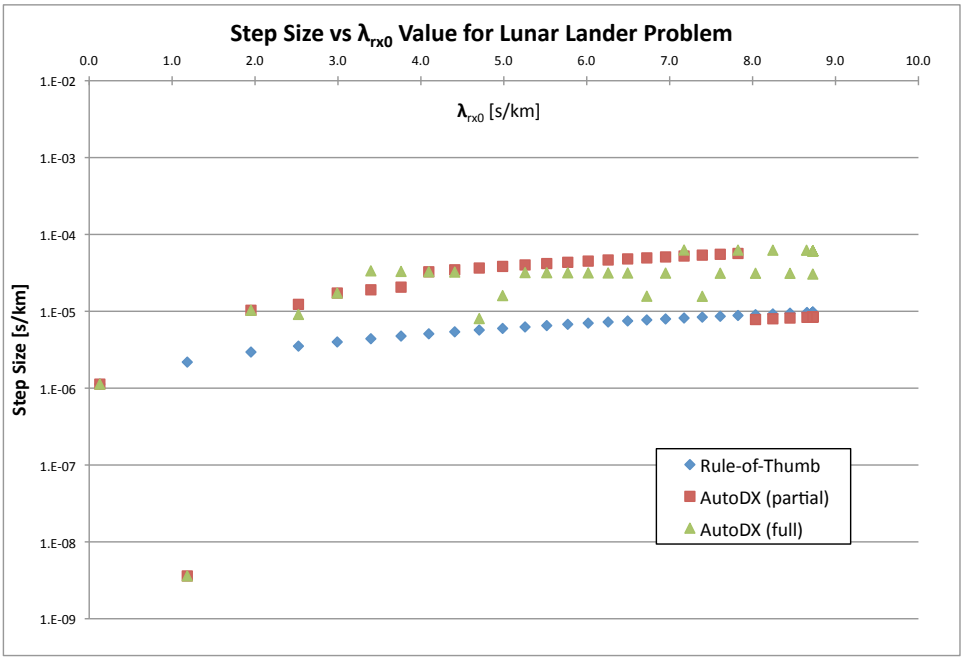


Figure 4.17: Step sizes computed by various methods for the optimization variable $\lambda_{rx0}$ of the Lunar Lander problem from example 4.8.

Figure 4.17 shows the step sizes for the variable $\lambda_{rx0}$ (the first element of $\mathbf{x}_p$) as it evolves from its initial guess value to the converged value over the course of the optimization. For each step-size estimation method, the achieved optimization convergence tolerance was $1e{-}12$. The rule-of-thumb method converged in 377 function evaluations. The first variation of AutoDX (used at every optimization iteration) converged in 15081 function evaluations. The second variation of AutoDX (used only when $\mathbf{x}_p$ changes considerably) converged in 2727 function evaluations.

The results in Figure 4.17 indicate that for this problem, the optimal step size is almost an order of magnitude greater than the rule-of-thumb step size. The partial use of AutoDX is seen to sufficiently match the results of fully using AutoDX, and has the benefit of considerably fewer function evaluations. Because the Lunar Lander problem is not particularly difficult to solve numerically, each method results in convergence. This is not necessarily true for all problems; for more complex problems, the partial-use case of AutoDX is usually desirable for its balance between an optimal step size and a reasonable number of function evaluations.

# Chapter 5

# Future Work and Conclusions

## 5.1 Optimized Algorithms

The step-size analysis theories presented in this dissertation give rise to multiple possible implementations. The algorithm used to create the examples, AutoDX, implements all of the basic theories but omits potential optimizations. This is done intentionally, in order to better observe truncation and roundoff error trends for various families of functions. Additional optimizations are now discussed, which forego a rigorous analysis of all possible step sizes in favor of potentially reduced function evaluations. Although these optimizations have not been implemented within AutoDX, it would be of great benefit for them to be included in any future implementations of the step-size analysis theory.

### 5.1.1 Store the Maximum Safe Step Size

The most straightforward optimization involves storing the maximum safe step size computed by AutoDX. On the next call to AutoDX, which may occur after several iterations of the optimization loop, the stored maximum safe step size is used as the initial large step size. The benefit of this simple optimization is that, for subsequent calls to AutoDX, many fewer function evaluations are required to find the valid truncation error region. For the Lunar Lander problem of example

4.8, this optimization results in a 5% reduction of total function evaluations. It should be noted that this optimization is not in the algorithm itself, but rather is a change in the arguments passed into the algorithm from the outer optimization loop. Pseudocode for this optimization is as follows.

```
dX_maxstep = 1.0 + abs(X)    // Initialize maximum tested step size
do optimization loop {
  call AutoDX(X, ... , dX_maxstep, ..., dX_maxsafe)
  if(no errors from AutoDX) {
    dX_maxstep = dX_maxsafe
  }
  ... // Remainder of optimization loop
}
```

### 5.1.2 Skipping the Valid Truncation Error Region

The AutoDX algorithm, as used herein, operates in a very linear fashion. The initial large step size is monotonically decreased until roundoff error is detected. In this process, many step sizes are tested which lie in the valid truncation error region. However, these step sizes are never actually used, and so it is useful to develop a method by which they are skipped.

To do this, it is recognized that the slopes of the valid truncation error and roundoff error regions are known constants (Section 3.4). First, the algorithm determines that a valid truncation error step size has been found, which corresponding to the maximum safe step size. From this, the truncation error best-fit

line can immediately be computed. Next, a few step sizes corresponding to the roundoff error region are tested, and the roundoff error best-fit line is computed. Finally, the intersection of these two best-fit lines (on a log-log scale) determines the optimal step size.

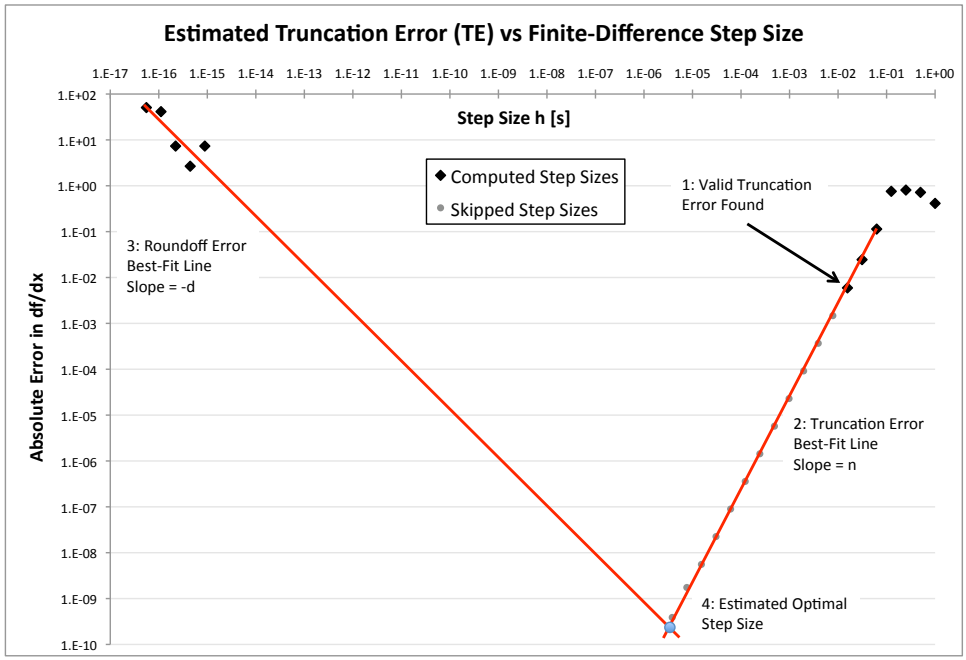


Figure 5.1: A more efficient step-size search algorithm, which skips analysis of many truncation error step sizes.

This process is illustrated in Figure 5.1. It is seen that by evaluating a few points in the roundoff error region, the algorithm can avoid computing many points in the truncation error region. Note that in practice, the roundoff error best-fit line would be computed using the corrected values of the roundoff error points from (2.78). This optimization is expected to reduce the number of function evaluations by 25% - 50%, depending on the size of the valid truncation

error region (which in turn depends on the function itself).

## 5.2 Numerical Integration

In addition to numerical differentiation, finite-difference equations are also commonly used in the design of numerical integration methods. In variable-step-size numerical integration, it is desired to use the *largest* possible step size in order to minimize the integration time. Therefore it is of interest to investigate how the theories developed in this dissertation could be used within the context of numerical integration.

## 5.3 Dissertation Conclusions

The original motivation for this research was to determine the optimal step size that minimizes errors in finite-difference derivative (FDD) computations. As is often the case, the pursuit of a seemingly simple solution led to a thorough understanding of a much more general and complicated problem. The research detailed in this dissertation presents a rigorous analysis of the analytical and numerical properties of (FDD) equations. Once the general FDD properties are understood, it becomes relatively simple to find specific properties such as an optimal step size.

The first chapter of this dissertation introduced the problem of determining an optimal step size, called the Step Size Dilemma. A history of FDD methods was given, which sheds light on previous efforts to solve this problem. These efforts

mainly focus on finding a step size which is optimal under certain assumptions of the function itself. In addition to FDD methods, a history of complex-step analysis and Automatic Differentiation was also presented. It was argued that although these methods are more modern and can produce far more accurate derivatives, the need for fully understanding finite-difference derivatives is still very relevant in today's world of simulation design.

The second chapter develops all analytical tools required to perform a full analysis of finite-difference derivative equations. Richardson Extrapolation, a method by which multiple low-order approximations are combined to produce a higher-order approximation, is introduced. It is then shown that all FDD equations can be represented in a common form, dependent on which derivative $d$ is approximated and the order of the error term $n$. This error term, called truncation error and expressed in the Lagrange Remainder form, depends in part on the step size $h$. Using a variation of Richardson Extrapolation, the truncation error term can be approximated by computing the finite-difference derivative for two successive step sizes.

From a purely mathematical standpoint, the error term for a FDD goes to zero with the step-size. However, it is shown that in practice there are two additional errors which play a very important role in step-size analysis. The first of these is brought on by the fact that the implementation of a function will accumulate errors internally. The more complex a function's implementation, the greater the chance that more errors will affect its output. This error is known as condition error, and is related to (but not the same as) the function's mathemat-

ical condition number. The second error arises from the subtraction operations inherent to every finite-difference method. When the step size is small, perturbed function values will be very close together and a computable amount of precision will be lost in the subtraction. This is called cancellation error, and it increases as the step size decreases. Together, condition and cancellation errors form an upper bound on the total roundoff error caused by finite-precision computation of a derivative using FDD methods.

The total error in a finite-difference derivative is shown to be bounded by the sum of roundoff and truncation errors. Since these errors grow in an inverse relation to each other, there must exist some step size for which their sum is minimized. A standard minimization is performed on the total error, and the result is an expression for the optimal step size in terms of the truncation, condition, and cancellation errors. Note that only the first two of these are unknown, since cancellation error is easily approximated.

Since a function's condition error and a FDD method's truncation error are in general unknown, the true error is also assumed to be unknown. However, it is shown that the truncation error approximation is a very good fit for the true error. For step sizes smaller than the optimal step size, the truncation error approximation must be corrected by a known constant factor. In addition, it is shown that the optimal step size computed using the estimated truncation error differs from the true optimal step size by another known constant.

The third chapter bridges the gap between the analytical formulation of the second chapter and the numerical analysis needed to develop a useable algorithm.

It is shown that the slope of the estimated truncation error function with respect to the step size is piecewise constant, when considered from a *log-log* perspective. For larger than optimal step sizes, the slope is equal to the order of the FDD method $n$. For step sizes smaller than the optimal, the truncation error estimates are greatly affected by roundoff error and therefore do not change smoothly. However, it is shown that a best-fit line through these points will have a slope of $-d$, where $d$ is the derivative being approximated. Furthermore, for step sizes *much* larger than the optimal, the slope is shown to be generally unpredictable.

Using the fact that the estimated truncation error slope is known, an algorithm is developed which iteratively seeks out the optimal step size. This algorithm is robust enough to skip over any initial step sizes which may be too large, and recognize when the region of predictable step size has been reached. The optimal step size obtained by this algorithm is easily adjusted to match the true optimal step size by using the correction factor derived in Chapter 2. This algorithm is shown to easily extend to multidimensional functions, while retaining memory efficiency and scalability.

Given an optimal step size, the condition error of a function can be approximated using theories derived in Chapter 2. Knowledge of this condition error can be invaluable in debugging a function implementation. If the condition error of a function is excessively high, then it would be prudent to consider whether inefficient algorithms are being used within the function to compute its output.

Finally, Chapter 3 identifies a few families of functions for which the step-size optimization theory will fail. These functions occur very rarely in real-world

use, and even if they do it is only for isolated sets of input variables. Nevertheless, should these deviant functions occur, methods are given by which they can be identified.

The step-size analysis theory and application given in Chapters 2 and 3 are implemented in a software package called AutoDX. Chapter 4 presents the results of using AutoDX to analyze several example functions, from simple polynomials to complex nonlinear optimization problems. For each example, derivatives are computed using AutoDX, a competing algorithm called GTA, and a rule-of-thumb step size. It is shown that AutoDX consistently computes a step size that is closest to the true optimal step size. In comparison, GTA often gets very close to the true optimal step size, and as expected the rule-of-thumb method is only close for well-conditioned functions.

In addition to the accuracy of the computed derivative, the performance of each step-size estimation method is compared. As expected, the rule-of-thumb method has by far the best performance with a total number of function evaluations proportional to the number of function inputs. In its current form, AutoDX is shown to require 10-40 times as many function evaluations as the rule-of-thumb method. With the optimizations outlined in this chapter, it is expected that AutoDX could perform with only 5-20 times as many function evaluations as the rule-of-thumb method. In comparison to these, the GTA algorithm can have 5-10 times as many function evaluations as AutoDX. This is caused in part by the fact that GTA does not employ concurrent analysis of a multidimensional function vector. In addition, because GTA is a statistical algorithm, it must necessarily

perform a large number of function evaluations to compile useful statistical data.

The author recognizes the large amount of preexisting work in the field of step-size analysis, which provided a great deal of guidance during the course of this research. It is the author's sincere hope that future researchers will be able to benefit from the formalization of step-size analysis theory presented herein.

# Appendices

# Appendix A

# The AutoDX Algorithm

The AutoDX algorithm, written in Fortran, implements most of the theories detailed in this dissertation. The version current as of this publication is outlined here.

```
subroutine ADXGetStepSize(X, n, dX_max, F_Fcn, m, iX, order,
                          writeoutput, dFdX_known, dX_out,
                          dFdX_out, dXmax_out, err_out, farerr_out)
```

This primary component of AutoDX computes the optimal step size for each element of a vector function F with respect to a particular element $i$ of an input vector X. These step sizes, when used with a particular finite-difference derivative equation, produce the least error in each element of $dF/dX_i$.

**Inputs:**

**X** (Vector) The independent variables.

**n** (Integer) Size of X.

**dX_max** (Vector) Maximum tested step size for each element of X.

**F_Fcn** (Subroutine) The function to differentiate. Defined as:

```
subroutine F_Fcn(X, n, F, m)
  integer, intent(in):: n, m
  double precision, intent(in):: X(n)
  double precision, intent(out):: F(m)
end subroutine F_Fcn
```

**m** (Integer) Size of output of F_Fcn.

**iX** (Integer) The derivative of F_Fcn is computed with respect to this element of X.

**order** (Integer) The truncation error order of the finite-difference equation used to compute derivatives.

**writeoutput** (Boolean) Whether analysis should be written to screen.

**dFdX_known** (Boolean Vector) Specifies elements of $dF/dX_i$ which are already known. Size m.

**Outputs:**

**dX_out** (Vector) The optimal step size for each element of F_Fcn. Size m.

**dFdX_out** (Vector) The gradient vector $dF/dX_i$. Size m.

**dXmax_out** (Vector) The maximum safe step size for each element of F_Fcn. Size m. This can be used to compute the maximum allowable change in X if a partial use of AutoDX is desired (Section 3.5.3).

**err_out** (Vector) Estimated relative error (roundoff + truncation) in $dF/dX_i$. Size m.

**farerr_out** (Vector) Condition error (Section 2.7.2) for each element of F_Fcn. Size m.

# Appendix B

# Finite-Difference Derivative Approximations

The most commonly used finite-difference derivative (FDD) approximations are given here, along with the various coefficients which are derived and used in this dissertation.

## B.1 FDD Approximations

As explained in Section 2.4, the general FDD equation is

$$FD_n^{(d)}(x,h) = \frac{\Delta f_n^{(d)}(x,h)}{h^d} + O(h^n) \tag{B.1}$$

where $d$ is the derivative order, $n$ is the truncation error order, and particular forms of $\Delta f_n^{(d)}$ are given in Table B.1. Here, the notation $f_i$ is used in place of $f(x+ih)$.

## B.2 Roundoff Errors

As the step size $h$ gets small, the perturbed function values from Table B.1 approach similar values, and roundoff errors become a significant factor in the FDD computation. The two types of roundoff error considered in this dissertation are Cancellation Error (Section 2.7.1) and Condition Error (Section 2.7.2).

Table B.1: Particular forms of finite-difference derivative equations.

| Type | $d$ | $n$ | $\Delta f_n^{(d)}(x,h)$ |
|---|---|---|---|
| Forward | 1 | 1 | $f_1 - f_0$ |
| Forward | 1 | 2 | $\frac{1}{2}\left(4f_1 - (f_2 + 3f_0)\right)$ |
| Backward | 1 | 1 | $f_0 - f_{-1}$ |
| Backward | 1 | 2 | $\frac{1}{2}\left((3f_0 + f_{-2}) - 4f_{-1}\right)$ |
| Central | 1 | 2 | $\frac{1}{2}\left(f_1 - f_{-1}\right)$ |
| Central | 1 | 4 | $\frac{1}{12}\left(8(f_1 - f_{-1}) + (f_{-2} - f_2)\right)$ |
| Central | 1 | 6 | $\frac{1}{60}\left(45(f_1 - f_{-1}) + 9(f_{-2} - f_2) + (f_3 - f_{-3})\right)$ |
| Forward | 2 | 1 | $(f_2 + f_0) - 2f_1$ |
| Central | 2 | 2 | $(f_1 + f_{-1}) - 2f_0$ |
| Central | 2 | 4 | $\frac{1}{12}\left(16(f_1 + f_{-1}) - (f_2 + f_{-2} + 30f_0)\right)$ |

As explained in Section 2.7.3, the total roundoff error is bounded by a combination of cancellation and condition errors.

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} \tag{B.2}$$

where $d$ is the derivative order. Particular forms of the error coefficients $|F_\epsilon|$ and $|F_\delta|$ are given in Tables B.2 and B.3, respectively. The notation $f_{\pm i}$ is used in place of $max(|f_i|, |f_{-i}|)$.

If the step size $h$ is assumed to be very small ($h < h_{\text{opt}}$), then it may be safe to assume that $f(x + ih) \approx f(x)$ for small values of $i$. In condensed notation, $f_i \approx f_0$. Under this assumption, the condition and cancellation error coefficient expressions can be simplified, and are given in Table B.4. Note that this assumption has not been fully tested for step sizes at or near $h_{\text{opt}}$.

Table B.2: Particular forms of the condition error coefficient.

| Type | $d$ | $n$ | $\lvert F_\epsilon \rvert$ |
|---|---|---|---|
| Forward | 1 | 1 | $\lvert f_1 \rvert + \lvert f_0 \rvert$ |
| Forward | 1 | 2 | $\frac{1}{2}\left(\lvert f_2 \rvert + 4\lvert f_1 \rvert + 3\lvert f_0 \rvert\right)$ |
| Backward | 1 | 1 | $\lvert f_0 \rvert + \lvert f_{-1} \rvert$ |
| Backward | 1 | 2 | $\frac{1}{2}\left(3\lvert f_0 \rvert + 4\lvert f_{-1} \rvert + \lvert f_{-2} \rvert\right)$ |
| Central | 1 | 2 | $\frac{1}{2}\left(\lvert f_1 \rvert + \lvert f_{-1} \rvert\right)$ |
| Central | 1 | 4 | $\frac{1}{12}\left(8(\lvert f_1 \rvert + \lvert f_{-1} \rvert) + (\lvert f_{-2} \rvert + \lvert f_2 \rvert)\right)$ |
| Central | 1 | 6 | $\frac{1}{60}\left(45(\lvert f_1 \rvert + \lvert f_{-1} \rvert) + 9(\lvert f_{-2} \rvert + \lvert f_2 \rvert) + (\lvert f_3 \rvert + \lvert f_{-3} \rvert)\right)$ |
| Forward | 2 | 1 | $\lvert f_2 \rvert + 2\lvert f_1 \rvert + \lvert f_0 \rvert$ |
| Central | 2 | 2 | $\lvert f_1 \rvert + 2\lvert f_0 \rvert + \lvert f_{-1} \rvert$ |
| Central | 2 | 4 | $\frac{1}{12}\left(16(\lvert f_1 \rvert + \lvert f_{-1} \rvert) + \lvert f_2 \rvert + \lvert f_{-2} \rvert + 30\lvert f_0 \rvert\right)$ |

Table B.3: Particular forms of the cancellation error coefficient.

| Type | $d$ | $n$ | $\lvert F_\delta \rvert$ |
|---|---|---|---|
| Forward | 1 | 1 | $max(\lvert f_1 \rvert, \lvert f_0 \rvert)$ |
| Forward | 1 | 2 | $\frac{1}{2}max(4\lvert f_1 \rvert, \lvert f_2 + 3f_0 \rvert)$ |
| Backward | 1 | 1 | $max(\lvert f_0 \rvert, \lvert f_{-1} \rvert)$ |
| Backward | 1 | 2 | $\frac{1}{2}max(\lvert 3f_0 + f_{-2} \rvert, 4\lvert f_{-1} \rvert)$ |
| Central | 1 | 2 | $\frac{1}{2}f_{\pm 1}$ |
| Central | 1 | 4 | $\frac{1}{12}\left(8f_{\pm 1} + f_{\pm 2}\right)$ |
| Central | 1 | 6 | $\frac{1}{60}\left(45f_{\pm 1} + 9f_{\pm 2} + f_{\pm 3}\right)$ |
| Forward | 2 | 1 | $max(\lvert f_2 + f_0 \rvert, 2\lvert f_1 \rvert)$ |
| Central | 2 | 2 | $max(\lvert f_1 + f_{-1} \rvert, 2\lvert f_0 \rvert)$ |
| Central | 2 | 4 | $\frac{1}{12}max(16\lvert f_1 + f_{-1} \rvert, \lvert f_2 + f_{-2} + 30f_0 \rvert)$ |

Table B.4: Simplified forms of the condition and cancellation error coefficients using small step sizes.

| Type | $d$ | $n$ | $\lvert F_\epsilon \rvert$ | $\lvert F_\delta \rvert$ |
|---|---|---|---|---|
| Forward | 1 | 1 | $2\lvert f_0 \rvert$ | $\lvert f_0 \rvert$ |
| Forward | 1 | 2 | $4\lvert f_0 \rvert$ | $2\lvert f_0 \rvert$ |
| Backward | 1 | 1 | $2\lvert f_0 \rvert$ | $\lvert f_0 \rvert$ |
| Backward | 1 | 2 | $4\lvert f_0 \rvert$ | $2\lvert f_0 \rvert$ |
| Central | 1 | 2 | $\lvert f_0 \rvert$ | $\frac{1}{2}\lvert f_0 \rvert$ |
| Central | 1 | 4 | $\frac{3}{2}\lvert f_0 \rvert$ | $\frac{3}{4}\lvert f_0 \rvert$ |
| Central | 1 | 6 | $\frac{11}{6}\lvert f_0 \rvert$ | $\frac{11}{12}\lvert f_0 \rvert$ |
| Forward | 2 | 1 | $4\lvert f_0 \rvert$ | $2\lvert f_0 \rvert$ |
| Central | 2 | 2 | $4\lvert f_0 \rvert$ | $2\lvert f_0 \rvert$ |
| Central | 2 | 4 | $\frac{16}{3}\lvert f_0 \rvert$ | $\frac{8}{3}\lvert f_0 \rvert$ |

## B.3 Optimal Step Size and Condition Error

The optimal step size can be estimated according to the derivation in Section 2.8. The equations for this, (2.66), is repeated here.

$$h_{\text{opt}} = \left[\frac{d}{n}\frac{1}{|C_n|}(\epsilon|F_\epsilon| + \delta|F_\delta|)\right]^{1/(n+d)} \tag{B.3}$$

Approximations of this equation for various FDD methods are given in Table B.5. The $|C_n|$ term is taken to be proportional to $|f^{(n+d)}(x)|$; the constant of proportionality is ignored. In addition, expressions for $|F_\epsilon|$ and $|F_\delta|$ are taken from Table B.4, and it is assumed that $\epsilon = \delta = 1e{-}16$. Under these assumptions, the approximations given closely resemble commonly accepted rules of thumb for the various FDD methods.

Table B.5: Approximations of the optimal step size for $f(x)$.

| Type | $d$ | $n$ | $h_{\text{opt}} \approx$ |
|---|---|---|---|
| Forward | 1 | 1 | $1e{-}8 \left\lvert \frac{f(x)}{f^{(2)}(x)} \right\rvert^{1/2}$ |
| Forward | 1 | 2 | $5e{-}6 \left\lvert \frac{f(x)}{f^{(3)}(x)} \right\rvert^{1/3}$ |
| Backward | 1 | 1 | $1e{-}8 \left\lvert \frac{f(x)}{f^{(2)}(x)} \right\rvert^{1/2}$ |
| Backward | 1 | 2 | $5e{-}6 \left\lvert \frac{f(x)}{f^{(3)}(x)} \right\rvert^{1/3}$ |
| Central | 1 | 2 | $1e{-}6 \left\lvert \frac{f(x)}{f^{(3)}(x)} \right\rvert^{1/3}$ |
| Central | 1 | 4 | $1e{-}4 \left\lvert \frac{f(x)}{f^{(5)}(x)} \right\rvert^{1/5}$ |
| Central | 1 | 6 | $1e{-}3 \left\lvert \frac{f(x)}{f^{(7)}(x)} \right\rvert^{1/7}$ |
| Forward | 2 | 1 | $1e{-}5 \left\lvert \frac{f(x)}{f^{(3)}(x)} \right\rvert^{1/3}$ |
| Central | 2 | 2 | $1e{-}4 \left\lvert \frac{f(x)}{f^{(4)}(x)} \right\rvert^{1/4}$ |
| Central | 2 | 4 | $1e{-}3 \left\lvert \frac{f(x)}{f^{(6)}(x)} \right\rvert^{1/6}$ |

# Bibliography


[1] Barton, R. P. Computing forward difference derivatives in engineering optimization. *Engineering Optimization 20* (1992), 205–224.

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

[3] Bean, W. C. Minimum $\Delta$V, three-impulse transfer onto a trans-mars asymptotic velocity vector. Tech. Rep. TN D-5757, NASA, April 1970.

[4] Brandt, T. Use of richardson extrapolation in error estimation of LES. In *Proceedings of the 19th Nordic Seminar on Computational Mechanics* (Lund, Sweden, 2006), pp. 143–146.

[5] Celik, I., Li, J., Hu, G., and Shaffer, C. Limitations of richardson extrapolation and some possible remedies. *Journal of Fluids Engineering 127* (July 2005), 795–805.

[6] Curtis, A., and Reid, J. The choice of step lengths when using differences to approximate jacobian matrices. *IMA Journal of Applied Mathematics 13*, 1 (1974), 121–126.

[7] Danby, J. The solution of kepler's equation. *Celestial Mechanics 40* (1987), 303–312.

[8] Edelbaum, T. N. Optimal nonplanar escape from circular orbits. *AIAA Journal 9*, 12 (1971), 2432–2436.

[9] Fornberg, B. Numerical differentiation of analytic functions. *ACM Transactions on Mathematical Software 7*, 4 (December 1981), 512–526.

[10] Gerbracht, R. J., and Penzo, P. A. Optimum three-impulse transfer between an elliptic orbit and a non-coplanar escape asymptote. In *AIAA/AAS Astrodynamics Specialist Conference Proceedings* (Jackson, WY, September 1968). AAS 68-084.

[11] Gibson, G. A. Taylor's theorem and Bernoulli's theorem: A historical note. In *Proceedings of the Edinburgh Mathematical Society* (January 2009), vol. 39, pp. 25–33. http://http://journals.cambridge.org.

[12] Gill, P., Murray, W., Saunders, M., and Wright, M. Computing forward-difference intervals for numerical optimization. *SIAM Journal of Scientific and Statistical Computing 4* (1983), 310–321.

[13] Gill, P., Murray, W., and Wright, M. *Practical Optimization*. Academic Press, London and New York, 1981.

[14] Gobetz, F., and Doll, J. A survey of impulsive trajectories, final report. Tech. Rep. G-910557-11, United Aircraft Research Laboratories, East Hartford, CT, June 1968.

[15] Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys 23* (1991), 5–48.

[16] Goodyear, W. H. Completely general closed-form solution for coordinates and partial derivatives of the two-body problem. *Astronomical Journal 70*, 3 (1965), 189–192.

[17] Goodyear, W. H. A general method for the computation of cartesian coordinates and partial derivatives of the two-body problem. Tech. Rep. CR-522, NASA, 1966.

[18] Greenberg, M. D. *Foundations of Applied Mathematics*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

[19] Hamming, R. W. *Numerical Methods for Scientists and Engineers*, 2nd ed. McGraw-Hill, Inc., New York, NY, 1973.

[20] Hoffman, J. D. *Numerical Methods for Engineers and Scientists*, 2nd ed. Marcel Dikker, Inc., New York, NY, 2001.

[21] Jones, D., and Ocampo, C. Optimal impulsive escape trajectories from a circular orbit to a hyperbolic excess velocity vector. In *AIAA/AAS Astrodynamics Specialist Conference Proceedings* (2010). AIAA Paper 2010-7524.

[22] Kahan, W. IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE 754, May 1996.

[23] Kedem, G. Automatic differentiation of computer programs. *ACM Transactions on Mathematical Software 6*, 2 (June 1980), 150–165.

[24] Lagrange, J. L. *Théorie des Fonctions Analytiques*, 1st ed. Public Domain, Paris, 1797. http://books.google.com.

[25] Lai, K. L. *Generalizations of the complex-step derivative approximation.* PhD thesis, University at Buffalo, Buffalo, NY, September 2006.

[26] Lantoine, G., Russell, R. P., and Dargent, T. Using multicomplex variables for automatic computation of high-order derivatives. In *AAS/AIAA Space Flight Mechanics Meeting* (San Diego, CA, February 2010). AAS 10-218.

[27] Lantoine, G., Russell, R. P., and Dargent, T. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software 38*, 3 (April 2012).

[28] Lyness, J. N. Differentiation formulas for analytic functions. *SIAM Journal on Numerical Analysis 22* (April 1968), 352–362.

[29] Lyness, J. N., and Moler, C. B. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis 4* (June 1967), 202–210.

[30] Martins, J. A., Kroo, I. M., and Alonso, J. J. An automated method for sensitivity analysis using complex variables. In *Proceedings of the 38th Aerospace Sciences Meeting* (Reno, NV, January 2000). AIAA Paper 2000-0689.

[31] Martins, J. A., Sturdza, P., and Alonso, J. J. The connection between the complex-step derivative approximation and algorithmic differentiation. In *Proceedings of the 38th Aerospace Sciences Meeting* (Reno, NV, January 2001). AIAA Paper 2001-0921.

[32] Martins, J. A., Sturdza, P., and Alonso, J. J. The complex-step derivative approximation. *ACM Transactions on Mathematical Software 29*, 3 (September 2003), 245–262.

[33] Ocampo, C., and Mathur, R. Variational model for optimization of finite-burn escape trajectories using a direct method. *Journal of Guidance, Control, and Dynamics 35*, 2 (March–April 2012).

[34] Ocampo, C., and Munoz, J.-P. Variational model for the optimization of constrained finite-burn escape sequences. In *AAS/AIAA Astrodynamics Specialst Conference Proceedings* (Pittsburth, PA, August 2009). AAS 09-381.

[35] Ocampo, C., and Munoz, J.-P. Variational equations for a generalized spacecraft trajectory model. *Journal of Guidance, Control, and Dynamics 33*, 5 (September - October 2010).

[36] Ocampo, C., and Saudemont, R. Initial trajectory model for a multi-maneuver moon-to-earth abort sequence. *Journal of Guidance, Control, and Dynamics 33*, 4 (2010), 1184–1194.

[37] Pemba, J.-P. Numerical differentiation by iterated complex perturbations. Tech. Rep. ISSN: 1933-1746, Prarie View A&M University, Dept. of Mathematics, Prarie View, TX, September 2006.

[38] Prentice, J. Truncation and roundoff errors in three-point approximations of first and second derivatives. *Applied Mathematics and Computation 217*, 9 (January 2011), 4576–4581.

[39] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[40] Pugh, R. E. A language for nonlinear programming problems. *Mathematical Programming 2* (1972), 176–206.

[41] Reiter, A., and Gray, J. H. A compiler of differentiable expressions (CODEX) for the CDC 3600. MRC Technical Summary Report 791, Mathematics Research Center, University of Wisconsin - Madison, December 1967.

[42] Restrepo, R. GTA: Package specification. Available on request, October 2011.

[43] Richardson, L. F. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philos. Trans. R. Soc. London 210*, A (1910), 307–357.

[44] Richardson, L. F., and Gaunt, J. A. The deferred approach to the limit. *Philos. Trans. R. Soc. London 226*, A (1927), 299–361.

[45] Robinson, S. B., and Geller, D. K. A simple targeting procedure for lunar trans-earth injection. In *AIAA Guidance Navigation and Control Conference Proceedings* (Chicago, IL, August 2009). AIAA.

[46] Salkuyeh, D. K. Stepsize control of the finite difference method for solving ordinary differential equations. *International Journal of Mathematical and Computer Sciences 5*, 4 (2009), 234–238.

[47] Science and Technology Facilities Council. HSL mathematical software library. http://www.hsl.rl.ac.uk/, 2012.

[48] Severance, C. IEEE 754: An interview with william kahan. *IEEE Computer 31*, 3 (March 1998), 114–115.

[49] Skeel, R. Roundoff error and the patriot missile. *SIAM News 25*, 4 (July 1992), 11.

[50] Squire, W., and Trapp, G. Using complex variables to estimate derivatives of real functions. *SIAM Review 40*, 1 (March 1998), 110–112.

[51] Stepleman, R. S., and Winarsky, N. D. Adaptive numerical differentiation. *Mathematics of Computation 33*, 148 (October 1979), 1257–1264.

[52] Taylor, B. *Methodus Incrementorum Directa & Inversa.* Public Domain, London, 1717. http://books.google.com.

[53] Trefethen, L. N., and Bau, D. *Numerical Linear Algebra.* SIAM, 1997.

[54] Warner, D. D. A partial derivative generator. Computing Science Technical Report 28, Bell Laboratories, Murray Hill, N.J., April 1975.

[55] Wertz, H. J. SUPER-CODEX: Analytic differentiation of FORTRAN statements. Aerospace Technical Report TOR-0172(9320)-12, The Aerospace Corporation, Los Angeles, CA, June 1972.

[56] Whitley, R., Ocampo, C., and Williams, J. Implementation of an autonomous multi-maneuver targeting sequence for lunar trans-earth injection. In *AIAA/AAS Astrodynamics Specialist Conference Proceedings* (2010). AIAA Paper 2010-8066.

[57] Williams, J., Davis, E., Lee, D., Condon, G., Dawn, T., and Qu, M. Global performance characterization of the three burn trans-earth injection maneuver sequence over the lunar nodal cycle. In *AIAA Astrodynamics Specialist Conference Proceedings* (August 2009).

[58] Yang, W. Y., Cao, W., Chung, T.-S., and Morris, J. *Applied Numerical Methods using MATLAB®*. John Wiley and Sons, Inc., Hoboken, NJ, 2005.