AAS 13-723

AN ANALYTICAL APPROACH TO COMPUTING STEP SIZES FOR FINITE-DIFFERENCE DERIVATIVES

Ravishankar Mathur*

Most numerical optimization methods require a derivative to determine search directions and magnitudes, and the prevailing method of computing these derivatives is the finite-difference approximation. While computationally straightforward, a finite-difference approximation requires estimation of a step-size parameter. An algorithm is presented here that computes the optimal step size for a finite-difference approximation of an unknown function's derivative. The algorithm's mathematical foundations are derived and numerical examples are given. The computed step size minimizes the combined roundoff and truncation errors in the finite-difference derivative, and the algorithm provides information on the validity of the step size with respect to changes in the independent variables. It is shown that the computed step size is correlated to the true optimal step size by a closed-form equation. The algorithm is also able to compute the function's condition error without additional user input.

INTRODUCTION

The problem of numerically determining the derivative of a function has been given an immense amount of attention from many subfields of science and engineering. Many algorithms have emerged that compute the derivative with varying levels of accuracy, generality, computation speed, and implementation complexity. These algorithms can be classified as either analytical, algorithmic, or finite-difference.

Analytical methods are those in which the derivatives are derived specifically for a particular class of problems. They produce exact solutions, but the time required to derive, implement, and validate the derivatives can be prohibitively high. In addition, computation of analytical derivatives can be considerably slower for complicated problems than any of the other methods. Finally, if the mathematical representation of the problem changes, the entire process of deriving, implementing, and validating the derivatives must be repeated. A prime example of the pros and cons of analytical differentiation algorithms is the Variational Model by Ocampo et al.,[1–3] which computes partial derivatives for multi-segment spacecraft trajectories. The process of deriving the equations for general trajectories and then implementing those equations in a particular programming language takes many months, and debugging the implemented code takes several more months. However, the end result is a function that produces derivatives accurate to almost full precision for trajectories with long times of flight.

Algorithmic differentiation, also known as automatic differentiation, involves analyzing a function's implementation and then differentiating each instruction within the function successively.[4–8] These methods have near-analytical accuracy and acceptable runtimes, but require an intimate

*Sr. GN&C Engineer, Emergent Space Technologies, Greenbelt MD. AIAA Member.

analysis of the function being differentiated. For compiled languages such as C/C++ or Fortran, complex-step differentiation requires a rigorous modification of the function code itself, a task which can be prohibitive for complicated functions or for large projects. When applied using complex numbers, algorithmic differentiation is closely related to the complex-step differentiation method,[9–16] which allows near-analytical computation of numerical derivatives using perturbations in the complex plane without the precision loss caused by differencing.

The most common numerical differentiation algorithms are the family of finite-difference algorithms. A finite-difference algorithm for computing a derivative is fast and easy to implement, has a very low runtime, and is completely independent of the function on which it operates. The only input required is a parameter associated with the independent variable of differentiation, known as the perturbation step size. There are guidelines as to what values can be used for this parameter, which makes the algorithm very simple to implement and use. Because of this ease of use, finite-difference algorithms have become the de facto accepted methods for numerically computing derivatives.

The tradeoff for this ease of use is reduced accuracy in the computed derivatives, as compared to the other derivative computation methods. This loss of accuracy is twofold; first due to mathematical truncation error in the finite-difference formulation, and second due to numerical roundoff error inherent to every finite-precision computation. In addition, a sufficiently incorrect choice of the perturbation step size can exacerbate either of these errors to the point where the computed derivative is too inaccurate to be of any use. When determining the gradient of multiple functions with respect to multiple independent variables, it quickly becomes difficult to estimate the optimal perturbation step size for each variable. Here, the optimal step size is defined as the one which, for a given variable, minimizes the sum of truncation and roundoff errors in the derivative of each function with respect to that variable. Complicating this problem further is the fact that as the value of the variable changes significantly, the optimal perturbation step size may also change.

The importance of correctly estimating the perturbation step size is seen by optimizing a three-impulse transfer from a Moon-centered orbit to a specified $\mathbf{V}_\infty$ vector.[17] 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 that such a maneuver often outperforms a single-impulse transfer,[18] 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. Great strides have been made in efficiently computing initial guess solutions for the three-impulse transfer, which helps to increase the chance of successful optimization.[19,20] However, due to the sheer amount of time spent finding step sizes (especially after the impulses were converted to finite burns), researchers have resorted to alternate specialized method just to compute the derivatives of the problem.[1,3]

The challenge of determining a step size that balances truncation and roundoff errors, sometimes referred to as the 'step-size dilemma', has received some academic attention. The Richardson Extrapolation method combines multiple low-order finite-difference approximations using successively decreasing step sizes to produce a single high-order approximation.[21,22] Alternatively, if high-order derivatives and roundoff errors of the function are known, then methods exist to estimate the optimal step size.[23,24] For simple analytical functions, such as combinations of polynomials or trigonometric functions, the roundoff error is often equal to the precision of the machine. However,

for complicated user-defined functions, it is nearly impossible to know the function's internal errors beforehand. Due to this limitation, the solutions given to the step-size dilemma have been deemed as useful only for trivial functions, and only in an academic sense.[25]

This study, taken from research presented in the author's doctoral dissertation,[26] approaches the step-size dilemma with the assumption that nothing is known about the implementation of the function being differentiated. Accurate expressions for both roundoff and truncation errors are derived, in a similar fashion as is done in previous research.[24] However, this study further analyzes both of these errors, and an algorithm is created that iteratively determines the optimal step size to minimize the total error in the derivative. This algorithm has several benefits over existing step-size estimation algorithms and rules of thumb, including:

- The function's condition error is estimated. This provides a measure of how much error accumulates within the function implementation itself, and can be useful for function debugging.
- No initial guess is required for the maximum possible step size.
- The algorithm can be used with any finite-difference approximation method, including the common forward- and central-difference methods.
- The algorithm determines the maximum allowable change in the independent variable for which the computed optimal step size is valid. Using this validity range, the algorithm need only be called when necessary.
- Multidimensional functions are handled intelligently, with no extra function calls.

PROBLEM STATEMENT

Given a scalar function f of a scalar independent variable x, the derivative of the function with respect to a particular value of x is formally defined as

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1}$$

where h is a perturbation step-size for x. For nontrivial functions, such a limit process cannot be directly performed, and so a Taylor Series formulation is employed. Although there are many such formulations, the simplest one proceeds as follows.

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{3!}f^{(3)}(x)h^3 + \cdots \tag{2}$$

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2}f''(x)h - \frac{1}{3!}f^{(3)}(x)h^2 - \cdots \tag{3}$$

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \tag{4}$$

where $O(h)$ encompasses all higher order terms. This equation, known as the forward-difference derivative approximation, differs from Eq.(1) in that the step-size h is treated as a parameter instead of a limiting index.

If the higher-order terms are ignored, then Eq.(4) becomes an approximation to the true derivative, with an error $O(h)$ proportional to the chosen h perturbation. A cursory examination of the $O(h)$

terms shows that $O(h) \rightarrow 0$ as $h \rightarrow 0$, which implies that h should be made as small as possible to maximize the accuracy of the approximation.

However, a more in-depth analysis of Eq.(4) 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.[27] 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 finite-difference operation in Eq.(4). 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.

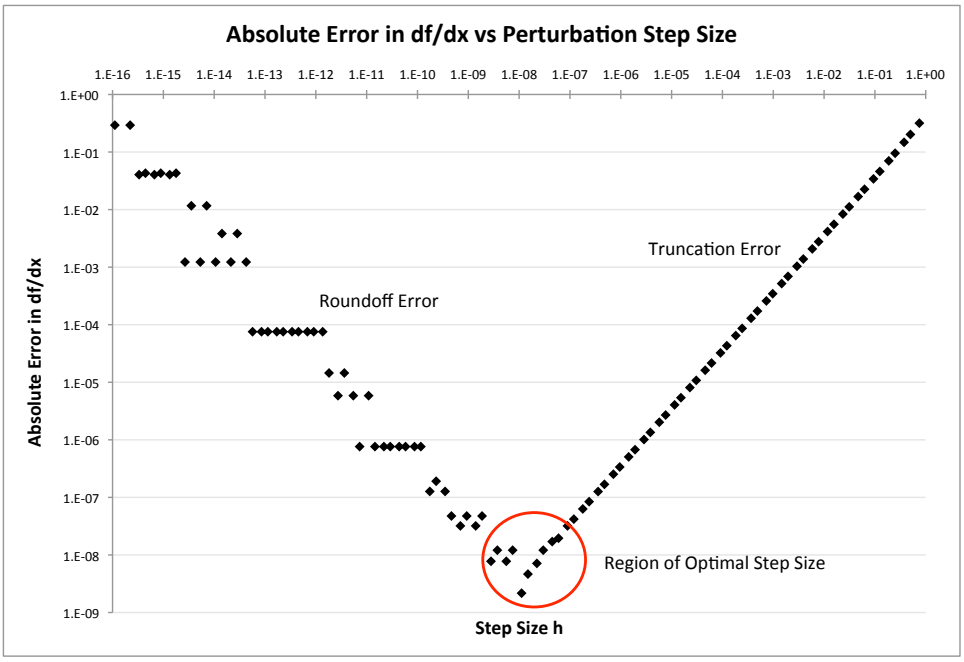

Figure 1. Absolute error in $f(x) = sin(x)$ with $x = \pi/4$, showing the step-size dilemma. The $O(h)$ forward-difference derivative approximation is used. The goal is to determine h such that the error in $f'(x)$ is minimized.

The classical step-size dilemma can be seen by plotting the error in the finite-difference derivative (FDD) as a function of the step size h, as illustrated in Figure 1. 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 the truncation error in $f'(x)$ increases, but roundoff error is virtually nonexistent. The optimal choice of h depends on requirements of the quality of the derivative, as discussed in the results section of this study.

ERROR FORMULATION

Truncation Error

The higher-order remainder terms $O(h)$ in the forward-difference approximation of Eq.(4) give the exact error associated with a particular value of the step-size h. These terms are generally not

known, because they involve the higher-order derivatives of $f(x)$. However, they can be consolidated by using the Lagrange form of the Taylor series remainder,[28]

$$O(h) = \frac{-1}{2} f''(\xi) h \tag{5}$$

$$x < \xi < x + h \tag{6}$$

where the exact value of ξ is undetermined. As $h \to 0$, ξ approaches x and the coefficient of h in the remainder approaches a limit,

$$\lim_{h \to 0} O(h) = \frac{-1}{2} f''(x) h \tag{7}$$

which is the expected truncation error in a forward difference approximation using a sufficiently small step size h, assuming that $f''(x)$ is smooth and bounded in the neighborhood of x.

A more accurate FDD approximation is the central-difference approximation. Using the Lagrange form of the Taylor series remainder term, the 2^{nd} order central difference approximation is given as

$$f(x + h) = f(x) + f'(x)h + \frac{1}{2} f''(x) h^2 + \frac{1}{3!} f^{(3)}(\xi^+) h^3 \tag{8}$$

$$f(x - h) = f(x) - f'(x)h + \frac{1}{2} f''(x) h^2 - \frac{1}{3!} f^{(3)}(\xi^-) h^3 \tag{9}$$

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} + O(h^2) \tag{10}$$

where the Intermediate Value Theorem is used to combine the two remainder terms,

$$O(h^2) = \frac{-1}{3!} f^{(3)}(\xi) h^2 \tag{11}$$

$$x < \xi^+ < x + h \tag{12}$$

$$x - h < \xi^- < x \tag{13}$$

$$x - h < \xi < x + h \tag{14}$$

Again, the exact values of ξ^+, ξ^-, and ξ are undetermined. The expected truncation error for sufficiently small h values, assuming that $f^{(3)}(x)$ is smooth and bounded in the neighborhood of x, is,

$$\lim_{h \to 0} O(h^2) = \frac{-1}{3!} f^{(3)}(x) h^2 \tag{15}$$

Because each finite-difference approximation has a unique truncation error equation 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,

$$f^{(d)}(x) = FD_n^{(d)}(x, h) + C(x, h) h^n \tag{16}$$

where $f^{(d)}(x)$ is the unknown true d^{th} derivative, $FD_n^{(d)}(x, h)$ is the particular finite-difference approximation (in the absence of roundoff errors), n is the order of the approximation, and $C(x, h)$ is the coefficient of the truncation error term. In Lagrange form,

$$C(x, h) = a_1 f^{(n+d)}(\xi) \tag{17}$$

$$x - a_2 h < \xi < x + a_3 h \tag{18}$$

where a_1, a_2, and a_3 are constants determined by the particular finite-difference approximation. Although $C(x,h)$ is undetermined (because it involves the unknown ξ), it can be approximated by using two step sizes h_1 and h_2, where $0 < h_2 < h_1$,

$$f^{(d)}(x) = FD_n^{(d)}(x,h_1) + C(x,h_1)h_1^n \tag{19}$$

$$f^{(d)}(x) = FD_n^{(d)}(x,h_2) + C(x,h_2)h_2^n \tag{20}$$

If h_1 and h_2 are both small and close in value, then $C(x,h_1) \approx C(x,h_2)$. A similar limiting process as that used in Eqs.(7) and (15) is used to define a new coefficient, $C_n(x)$, which is independent of the step size and approximates the unknown coefficients,

$$C_n(x) \approx C(x,h_1) \approx C(x,h_2) \approx a_1 f^{(n+d)}(x) \tag{21}$$

The truncation error for the finite-difference equation is related to this approximation by,

$$\lim_{h\to 0} O(h^n) = a_1 f^{(n+d)}(x)h^n \approx C_n(x)h^n \tag{22}$$

With the understanding that all functions are evaluated at x with appropriate step sizes h_1 and h_2, the two finite-difference equations are rewritten with reduced notation,

$$f^{(d)} = FD_1^{(d)} + C_n h_1^n \tag{23}$$

$$f^{(d)} = FD_2^{(d)} + C_n h_2^n \tag{24}$$

These are two linear equations in two unknowns, $f^{(d)}$ and C_n. They are solved for C_n, yielding

$$C_n = \frac{FD_2 - FD_1}{h_1^n - h_2^n} \tag{25}$$

With this computed value of C_n, an estimate of the truncation error for the FDD approximation of order n using a step size h_1 is,

$$TE_n(x,h_1) = C_n(x)h_1^n \tag{26}$$

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 $h_1 > h_2$. Furthermore, this derivation of the truncation error estimate is almost identical to the initial steps of a Richardson extrapolation procedure.[22] However, the goal of Richardson extrapolation is to increase the order of the given finite-difference approximation, whereas the goal of this study is to obtain an error estimate for the given finite-difference approximation.

An example of the use of the estimated truncation error from Eqs.(25) and (26) is given in Figure 2. The similarities between this estimated truncation error and the absolute error from Figure 1 are clear. Both exhibit a decreasing truncation error until a limiting point is reached, followed by an increasing roundoff 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, for which the estimated error is zero. These points occur because the successive finite-difference approximations from Eq.(25) 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 that roundoff error dominates the computations. However, their existence does not lessen the

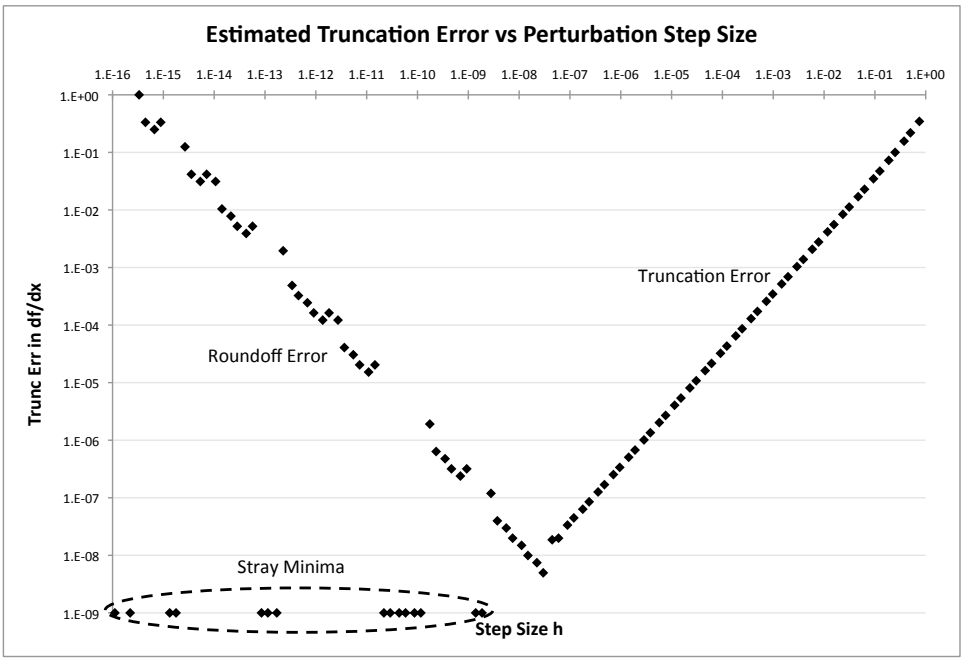

Figure 2. Estimated truncation error for $f(x) = sin(x)$ at $x = \pi/4$, using the $O(h)$ forward-difference derivative approximation. Error trends are preserved as compared to the true error in Figure 1.

effectiveness of using estimated truncation error plots to analyze error trends. Namely, roundoff error affects the estimated truncation error at nearly the same value of h at which it affects the true error. This fact provides a powerful analytical tool with which to determine the optimal step size that minimizes the total error of the chosen finite-difference approximation.

There is one important difference between the estimated truncation error and the true error that remains to be addressed. Namely, the step size h that minimizes the estimated truncation error is not exactly the same as the h that minimizes the true error. This is due to fact that the estimated truncation error is computed using two distinct step sizes. Given the step sizes used in Eq.(25), the step-size reduction ratio t can be defined as $t = h_2/h_1$. The relationship between the estimated optimal step size $h_{\text{opt,TE}}$ and the true optimal step size $h_{\text{opt,true}}$ is shown in Reference [26] to be,

$$h_{\text{opt,true}} = \left(\frac{1}{t^*}\right)^{1/(n+d)} h_{\text{opt,TE}} \tag{27}$$

$$t^* = \frac{1 + (1/t)^d}{1 - t^n} \tag{28}$$

This equation is critical because it bridges the gap between the estimated truncation error (which can be computed) and the true error (which usually cannot be computed) for a given FDD method. It is therefore confirmed that the estimated truncation error can in fact be used to construct an algorithm that finds the optimal step size.

Roundoff Error

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. Any finite-difference computation is subject to

two types of roundoff error: subtractive cancellation error and condition error. Given two numbers a and b represented in fixed precision, subtractive cancellation error occurs when a significant number of the leading digits of a and b are equal. An indicative example of subtractive cancellation error is seen by assuming that a and b are defined as,

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

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

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

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. An accurate upper bound to this error is,

$$|(a-b)_{\text{true}} - (a-b)| \leq \delta \max(|a|, |b|) \tag{29}$$

where δ 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 [REF??]), $\delta = 2^{-53}$. It is important to note that Eq.(29) only gives an upper bound on subtractive cancellation error; there is no way to know the exact error.

In contrast to subtractive cancellation error, condition error occurs within the implementation of the function $f(x)$ itself. The function may involve trigonometric calculations, numerical integrations, or any number of other mathematical operations when computing its result. Each of these operations contains a small amount of error, which accumulates throughout the function, resulting in a substantial error in the function result. Although this error cannot generally be predicted or exactly computed beforehand, its upper bound can be estimated. Assuming that condition error affects all digits in the result below a certain threshold, an approximation of the upper bound of the error is given as,

$$|f(x)_{\text{true}} - f(x)| \leq \epsilon |f(x)| \tag{30}$$

where ϵ indicates the magnitude of the most significant digit affected by condition error. Although ϵ is equal to machine precision for trivial and many built-in functions ($\sin$, $\cos$, etc...), in general it is an unknown to be computed. Conversely, once ϵ is computed, it can be of great assistance in determining the amount of condition error introduced by a given implementation of $f(x)$.

The roundoff error bounds in a finite-difference computation are formed using the expressions for subtractive cancellation and condition errors given in Eqs.(29) and (30). For the forward-difference derivative approximation given in Eq.(4), using condensed notation,

$$FD_1^{(1)}(x,h) = FD = \frac{f(x+h) - f(x)}{h} = \frac{f_1 - f_0}{h} \tag{31}$$

where subscripts for f are used to indicate the step size as a multiple of h.

Subtractive cancellation error is bounded by,

$$FD_{\text{true}} - FD = \frac{(f_1 - f_0)_{\text{true}} - (f_1 - f_0)}{h} \tag{32}$$

$$|FD_{\text{true}} - FD| \leq \frac{\delta |f_{1/0}|}{h} \tag{33}$$

$$|f_{1/0}| = \max(|f_1|, |f_0|) \tag{34}$$

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

Condition error is bounded by,

$$FD_{\text{true}} - FD = \frac{(f_{1,\text{true}} - f_1) - (f_{0,\text{true}} - f_0)}{h} \tag{35}$$

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon_1|f_1| + \epsilon_0|f_0|}{h} \tag{36}$$

If h is small enough such that the code paths taken by $f(x)$ and $f(x+h)$ are the same (or similar), then the magnitudes of the relative errors ϵ_0 and ϵ_1 introduced by both calls to f will be the approximately equal. In addition, for the worst case, the signs of the errors will be the same, causing the errors to compound. The condition error then simplifies to,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_0|)}{h} \tag{37}$$

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.

In the general case, both subtractive 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,

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_0|) + \delta|f_{1/0}|}{h} \tag{38}$$

where, in general, ϵ is unknown.

For the central-difference derivative approximation given in Eq.(10), using condensed notation,

$$FD = \frac{f_1 - f_{-1}}{2h} \tag{39}$$

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

$$|FD_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_{-1}|) + \delta|f_{\pm 1}|}{2h} \tag{40}$$

$$|f_{\pm 1}| = \max(|f_1|, |f_{-1}|) \tag{41}$$

Roundoff error bounds for higher order finite-difference approximations are given in the Appendix of Reference [26].

Total Error

To formulate the estimate for total error in a FDD, the general form of Eq.(16) is rewritten using condensed notation,

$$f_{\text{true}}^{(d)} = FD_{\text{true}} + C_{\text{true}} h^n \tag{42}$$

Subtracting the computed FDD, which contains roundoff errors, gives the total error in the FDD as compared to the true derivative,

$$f^{(d)}_{\text{true}} - FD = FD_{\text{true}} - FD + C_{\text{true}} h^n \tag{43}$$

where the true FDD, FD_{true}, is unknown. Taking the magnitude of the total error and applying the triangle inequality gives an upper bound on the total error,

$$|f^{(d)}_{\text{true}} - FD| = |FD_{\text{true}} - FD + C_{\text{true}} h^n| \tag{44}$$

$$\leq |FD_{\text{true}} - FD| + |C_{\text{true}} h^n| \tag{45}$$

If h is small enough such that Eq.(21) applies,

$$|f^{(d)}_{\text{true}} - FD| \leq \frac{\epsilon |F_\epsilon| + \delta |F_\delta|}{h^d} + |C_n| h^n \tag{46}$$

where the explicit expression for $|FD_{\text{true}} - FD|$ is given by Eq.(38), (40), or the appropriate round-off error bounding equation from the Appendix of Reference [26]. For the commonly used 2^{nd}-order central-difference approximation of the first derivative,

$$|f'_{\text{true}} - FD| \leq \frac{\epsilon(|f_1| + |f_{-1}|) + \delta |f_{\pm 1}|}{2h} + |C_2| h^2 \tag{47}$$

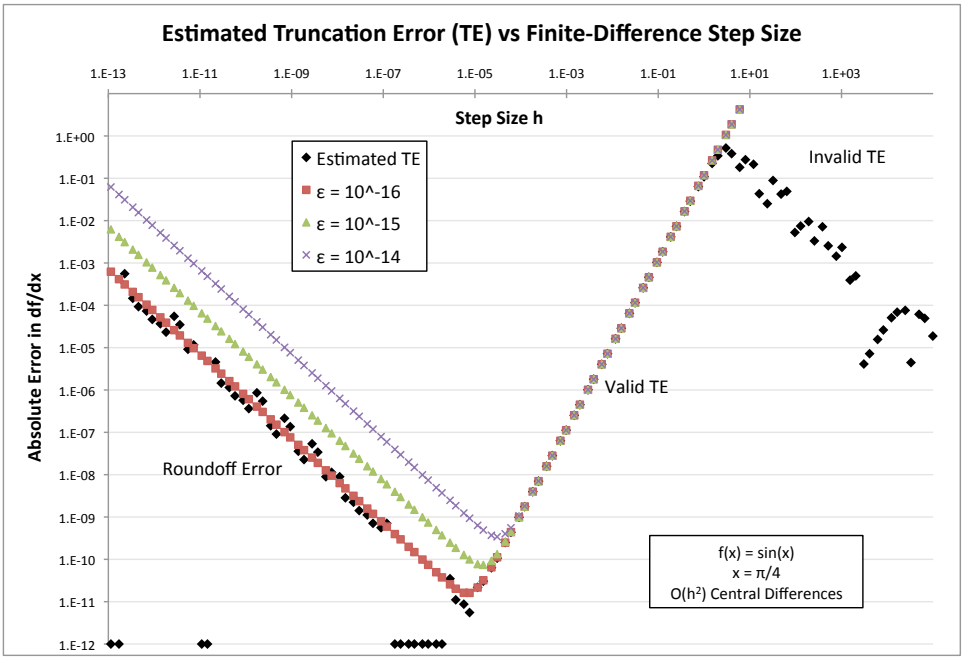

Figure 3. Effect of condition error ϵ on the total error bound.

An example of the effect of condition error ϵ is seen in Figure 3. For the function $f(x) = \sin(x)$ with $x = \pi/4$ and using the central-difference approximation, the total error bound from Eq.(47) is plotted for various values of ϵ, and the estimated truncation error from Eq.(26) is also plotted for comparison. C_2 is computed as $\frac{-1}{3!} f^{(3)}(x)$, and $\delta = 2^{-53}$ since double-precision numbers are used.

It can be seen that in the roundoff error range of step sizes, the ϵ 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 is not coincidental, and is used to develop an optimal step-size estimation algorithm.

Figure (3 also shows the effect of using step sizes large enough to violate the 'sufficiently small step size' assumptions made so far. For these step sizes, the computed estimated truncation error from Eq.(26) seem almost nonsensical, and certainly do not follow the predicted truncation or roundoff error patterns derived in this research. This fact is also used when developing an optimal step-size estimation algorithm.

Optimal Step Size Derivation

The optimal step size h_{opt} that minimizes the true total error of the finite-difference approximation also minimizes the error function $E(x, h)$, defined as the right-hand side of Eq.(46),

$$E(x,h) = \frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n|h^n \tag{48}$$

Minimizing $E(x, h)$ with respect to the step size yields the analytical solution for the optimal step size h_{opt} as a function of all parameters of interest:

$$h_{\text{opt}} = \left[\frac{d}{n}\frac{1}{|C_n|}(\epsilon|F_\epsilon| + \delta|F_\delta|)\right]^{1/(n+d)} \tag{49}$$

This equation takes into account the derivative of interest d, the order n of the finite-difference equation, the system's numerical precision δ, and the function's condition error ϵ. This equation simplifies to the step-size rule-of-thumb approximations of Gill et al.[29,30] for the case of the first derivative ($d = 1$) using the forward-difference approximation ($n = 1$).

If the condition error ϵ is not known, but the optimal step size h_{opt} can somehow be determined, then the condition error can be computed as,

$$\epsilon = \frac{1}{|F_\epsilon|}\left(\frac{n}{d}|C_n|h_{\text{opt}}^{n+d} - \delta|F_\delta|\right) \tag{50}$$

SIMPLE STEP SIZE ESTIMATION ALGORITHM

The total error function in Eq.(46) is analyzed from a logarithmic standpoint to reveal linear trends as the step size moves away from its optimal value.

$$\ln E(x,h) = \ln\left(\frac{\epsilon|F_\epsilon| + \delta|F_\delta|}{h^d} + |C_n|h^n\right) \tag{51}$$

$$\ln E(x,h) = \begin{cases} \ln\left(\epsilon|F_\epsilon| + \delta|F_\delta|\right) - d\ln h & h \ll h_{\text{opt}} \\ \\ \ln|C_n| + n\ln h & h \gg h_{\text{opt}} \end{cases} \tag{52}$$

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 earlier, the truncation error estimate from Eqs.(25) and (26) accurately models the true error when used with the correction factor in (27). Therefore, it is subject to the same linear trends regarding small and large step sizes. Furthermore, since the estimated truncation error is not a valid approximation to the true error for very large step sizes, it is expected that the linear trend (in the logarithmic sense) does not hold for these step sizes.

Given the theory derived and analyzed in preceding sections, an algorithm can now be developed that iteratively finds the optimal step size h_{opt} for a given FDD method. Although the algorithm is completely general, it is beneficial to develop it using the following example problem as a reference: A 2-body orbit about the Earth, with orbital parameters given in Table 1, is propagated for a quarter of its period from a fixed initial true anomaly of $\nu_0 = 0^\circ$. Danby's method* is used to solve Kepler's equation,[31] and the final result is the true anomaly ν_f at the final time t_f. Figure 4 shows the estimated truncation error when the derivative $d\nu_f/dt_f$ is computed using the second-order central-difference method.

Table 1. Orbital parameters for propagated orbit.

μ	$398600.4\,[km^3/s^2]$
a	$200000\,[km]$
e	0.96453
i	51.619°
ω, Ω, ν_0	0°
(t_0, t_f)	$(0, 222533.8)\,[s]$

A simple step-size optimization algorithm can be developed by visually analyzing Figure 4. The general goal of this algorithm is as follows: Start with a very large initial step size, ignore any initial invalid truncation errors, find the region of valid truncation errors, and stop when roundoff error overtakes truncation error.

1. A large initial step size h_0 is chosen, preferably one that is a power of 2 as explained in Reference [26]. 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 truncation error slope nearly equal to n by coincidence. These cases occur sporadically; it may happen 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 was shown 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.

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

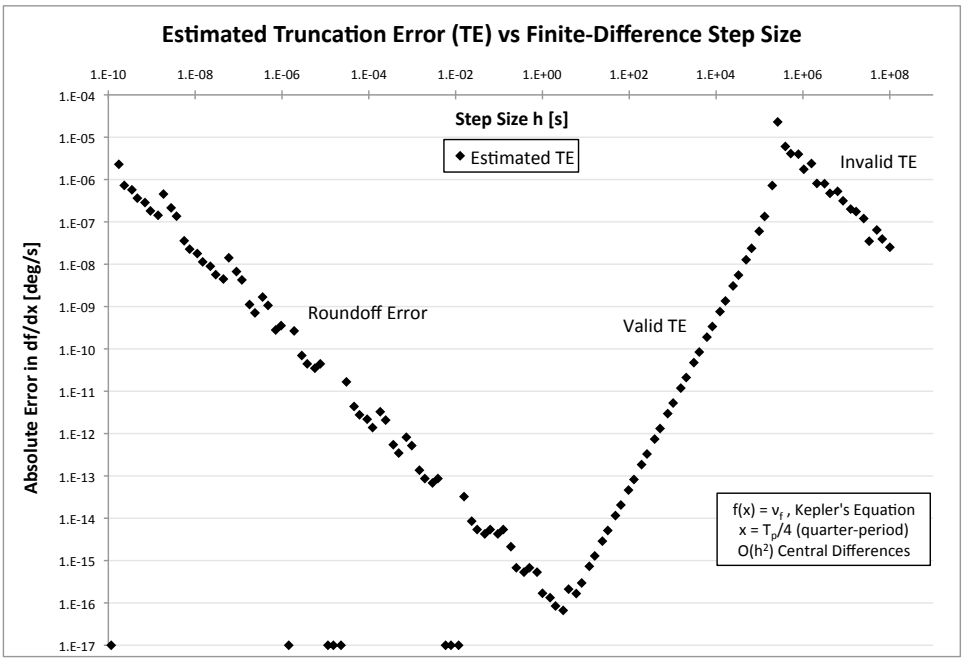

Figure 4. A truncation error plot used to develop the simple algorithm.

3. The current step size h_i and next step size $h_{i+1} = th_i$ are used with Eqs.(25) and (26) 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 (25) 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 (27): $h_{\text{opt,true}} = (t^*)^{-1/(n+d)} h_i$. Using this optimal step size and the previously saved C_n value, the condition error ϵ can be computed using (50). The algorithm exits with the optimal step size $h_{\text{opt,true}}$ and the computed condition error ϵ.

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 orbit propagation example in Figure 4, the solution computed by this algorithm is given in Table 2. Note that the corrected step size is adjusted to the closest power of 2 which has already been tested, to reduce function evaluations. In both corrected and uncorrected cases, the function condition error ϵ 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].

Table 2. Solution for Kepler orbit propagation problem.

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

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 constituent gradient vectors of the Jacobian matrix, 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 (with respect to function calls) of $O(nm)$, but this is easily reduced by noting that the function of interest 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 function's 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. All that remains is to determine which computed step size to use for any given input variable, since optimal step sizes are computed for each output variable. Various options for this decision are discussed in Reference [26].

EXAMPLES

An implementation of the simple step-size search algorithm called AutoDX has been developed, and is now used to determine the optimal step size for a variety of test functions. For each test case, the results of the AutoDX algorithm are compared to those obtained using two other methods: (1) using a rule-of-thumb step size and (2) using the Gradient Tuned Algorithm (GTA).[32] The GTA is a statistical step-size search algorithm that starts with a very small step size, and analyzes 'batches'

of FDD values at each step size up to some maximum. The algorithm stops when the dispersion of step sizes within each successive batch reaches a minimum.

Fundamenal 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. An example of this is the following periodic function

$$f(x) = \sin(x)\cos(3x) \tag{53}$$

This function 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 3, with the AutoDX results computed using the corrected optimal step size. The AutoDX algorithm determined that the maximum valid step size $h_{\text{max}} = 0.25$, and the condition error in $f(x)$ is less than machine precision.

Table 3. Optimal step size h_{opt} for computing derivative of $f(x) = \sin(x)\cos(3x)$.

	h_{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 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 5 is that the GTA optimal step size seems to have almost zero estimated error, but Table 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 cancellations. Such a repeated occurrence of nearly equal FDD values causes the C_n estimate from (25) – and therefore also the truncation error estimate from (26) – 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_{opt} (as approximated by the corrected AutoDX h_{opt}) should be avoided.

Algorithmic Functions

Algorithmic functions are those that implement a nontrivial algorithm to produce their results. They often employ iterative methods such as numerical integrators or solvers to compute a result. Because numerical errors can compound within these iterations, algorithmic functions are expected to have a greater condition error than simple fundamental functions. To showcase the performance of AutoDX for step-size optimization with algorithmic functions, a Lunar intercept problem is considered. Given an initial Earth-centered orbit, an optimal 2-body transfer to the Moon is computed.

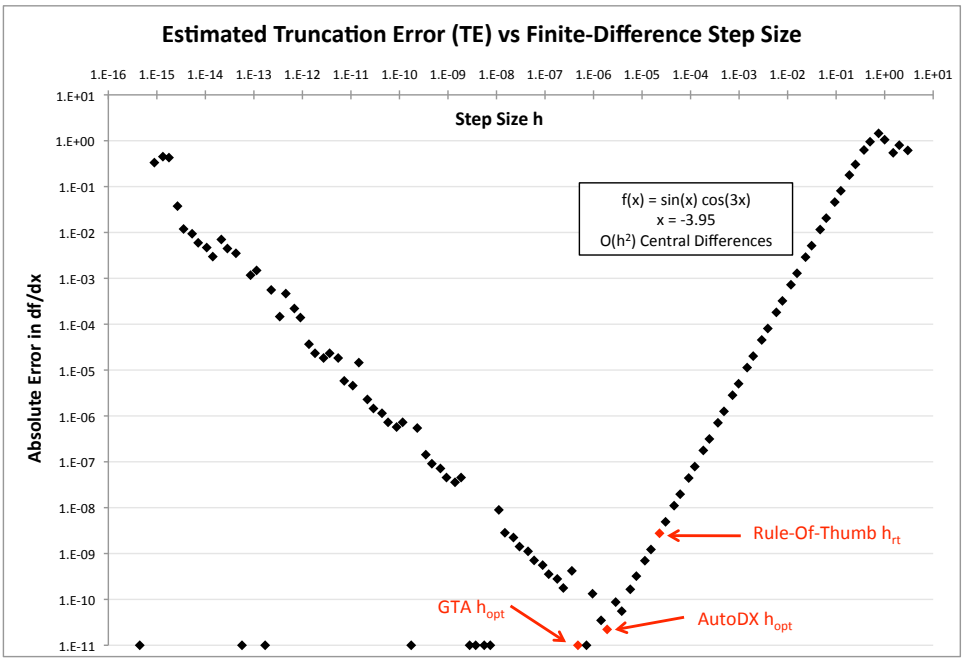

Figure 5. Estimated truncation errors for $f(x) = \sin(x)\cos(3x)$ example.

The initial and Lunar orbits (given in Table 4) are frozen, and the epoch time t_0, initial Lunar true anomaly $\nu_{0,Moon}$, and initial orbit true anomaly ν_0 are all specified. The satellite coasts on the initial orbit for time dt_1, after which an impulse $\Delta\mathbf{v}$ is applied to put the satellite on 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 6. The performance index for this problem is

$$J = \|\Delta\mathbf{v}\| = \Delta v \tag{54}$$

and the optimization variable and constraint vectors are

$$\mathbf{x}^\top = \begin{pmatrix} dt_1 & dt_{ga} & \Delta\mathbf{v}_x & \Delta\mathbf{v}_y & \Delta\mathbf{v}_z \end{pmatrix}_{1\times 5} \tag{55}$$

$$\mathbf{c} = \begin{pmatrix} \mathbf{r}_{\mathbf{Moon}}(t_f) - \mathbf{r}(t_f) = \mathbf{0} \end{pmatrix}_{3\times 1} \tag{56}$$

This example also shows the effectiveness of using AutoDX for multidimensional problems.

The optimization process drives $\mathbf{c}$ to zero while minimizing the total cost (Δv), and requires the gradients of both of these quantities with respect to the optimization variables $\mathbf{x}$. The latter gradient has a simple analytical solution, but the Jacobian of the constraint function $\dfrac{\partial \mathbf{c}}{\partial \mathbf{x}}$ is much more difficult to compute. Because this problem involves purely ballistic trajectory arcs, the state transition matrix approach of Goodyear[33,34] or the variational method of Ocampo and Munoz[1] 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.

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

$$\mathbf{x}_0^\top \approx \begin{pmatrix} 37391 & 414135 & \text{-}1.34 & \text{-}4.09 & 0.18 \end{pmatrix}_{1\times 5} \tag{57}$$

Table 4. Initial and Lunar orbits for the Lunar transfer example.

	Initial Orbit	Lunar Orbit
a	$24555.6[km]$	$384400.0[km]$
e	0.731921	0.0549
i	$51.619°$	$19.0°$
ω	$45.0°$	$0.0°$
Ω	$250.0°$	$0.0°$
ν_0	$0.0°$	$90.0°$

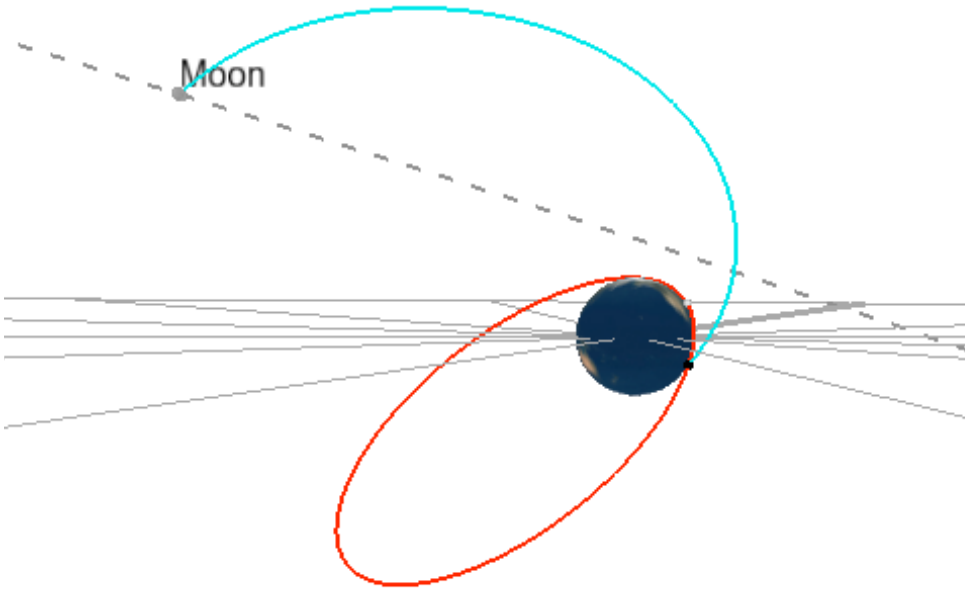

Figure 6. Setup of the Lunar transfer problem. The initial orbit is in red, and the transfer orbit is in blue.

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 previously, there may be a separate optimal step size h_{opt} associated with each partial derivative $\dfrac{\partial \mathbf{c}_j}{\partial \mathbf{x}_i}$. Table 5 gives each of these step sizes as computed by AutoDX and GTA, as well as the single rule-of-thumb step size for each $\mathbf{x}_i$.

It is clear that the optimal step sizes chosen by AutoDX and GTA are very similar for all elements of $\mathbf{x}$, but are several orders of magnitude removed from the rule-of-thumb step size h_{rt} for the first two elements (dt_1 and dt_{ga}). A more distinct contrast is seen when comparing the number of function evaluations used by each method, as shown in Table 6. As expected, using a fixed step size completely outperforms all other methods. GTA has a high number of function evaluations

Table 5. Optimal step sizes for each element of $\mathbf{x}$, given in $[s]$ for dt and $[km/s]$ for Δv.

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

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 for future invocations of AutoDX. In addition, Reference [26] discusses several optimizations that can be implemented within AutoDX to significantly reduce its computational cost.

Table 6. Number of function evaluations in computing $\dfrac{\partial \mathbf{c}}{\partial \mathbf{x}_i}$.

	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

CONCLUSIONS

The research presented here, and further detailed in the author's dissertation,[26] gives a rigorous mathematical analysis of step-size estimation theory. The tools derived through this analysis are used to estimate the error in a particular finite-difference derivative (FDD). In particular, it is shown that the FDD error follows very predictable patterns (on a log-log scale) for both large and small step sizes. For large step sizes, truncation error in the FDD approximation dominates and the slope

of the estimated FDD error with respect to the step size is equal to the order of the chosen FDD method. For small step sizes, roundoff error dominates and the best-fit slope is the negative of the derivative being estimated.

Combining information about the estimated truncation error yields a simple step-size estimation algorithm, which effectively seeks out the step size which minimizes the sum of roundoff and truncation errors. 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 a constant correction factor. In addition, this algorithm is shown to easily extend to multidimensional functions, while retaining memory efficiency and scalability.

It should be noted that this research provides two main benefits over existing step-size estimation algorithms. Firstly, it ties together the optimal step size with the condition error of the differentiated function's implementation. Having a condition error estimate can be shown to be extremely beneficial in verifying a function's accuracy. Secondly, this research not only provides an optimal step size, but also quantifies the validity of that step size with respect to changes in the independent variable. This allows a specific step size to be used within an encompassing optimization loop without having to re-run the algorithm at every optimization iteration.

Finally, it is recognized that significant optimizations can be made to the AutoDX algorithm. As it stands, AutoDX requires significantly more function evaluations as a rule-of-thumb method, but this can certainly be reduced with ongoing research.

ACKNOWLEDGMENTS

The author would like to thank Dr. Cesar Ocampo of UT Austin for his support of this research, and Dr. George Davis and Everett Cary of Emergent Space Technologies for their understanding of time requirements and flexibility with work schedules.

REFERENCES

[1] C. Ocampo and J.-P. Munoz, "Variational Equations for a Generalized Spacecraft Trajectory Model," *Journal of Guidance, Control, and Dynamics*, Vol. 33, September - October 2010.

[2] C. Ocampo and J.-P. Munoz, "Variational Model for the Optimization of Constrained Finite-Burn Escape Sequences," *AAS/AIAA Astrodynamics Specialst Conference Proceedings*, Pittsburth, PA, August 2009. AAS 09-381.

[3] C. Ocampo and R. Mathur, "Variational Model for Optimization of Finite-Burn Escape Trajectories Using a Direct Method," *Journal of Guidance, Control, and Dynamics*, Vol. 35, March–April 2012.

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

[5] R. E. Pugh, "A language for nonlinear programming problems," *Mathematical Programming*, Vol. 2, 1972, pp. 176–206.

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

[7] A. Reiter and J. H. Gray, "A Compiler of Differentiable Expressions (CODEX) for the CDC 3600," MRC Technical Summary Report 791, Mathematics Research Center, University of Wisconsin - Madison, December 1967.

[8] G. Kedem, "Automatic Differentiation of Computer Programs," *ACM Transactions on Mathematical Software*, Vol. 6, June 1980, pp. 150–165.

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

[10] G. Lantoine, R. P. Russell, and T. Dargent, "Using Multicomplex Variables for Automatic Computation of High-Order Derivatives," *ACM Transactions on Mathematical Software*, Vol. 38, April 2012.

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

[12] J. N. Lyness, "Differentiation Formulas for Analytic Functions," *SIAM Journal on Numerical Analysis*, Vol. 22, April 1968, pp. 352–362.

[13] W. Squire and G. Trapp, "Using Complex Variables to Estimate Derivatives of Real Functions," *SIAM Review*, Vol. 40, March 1998, pp. 110–112.

[14] J. A. Martins, P. Sturdza, and J. J. Alonso, "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, September 2003, pp. 245–262.

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

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

[17] R. Whitley, C. Ocampo, and J. Williams, "Implementation of an Autonomous Multi-Maneuver Targeting Sequence for Lunar Trans-Earth Injection," *AIAA/AAS Astrodynamics Specialist Conference Proceedings*, 2010. AIAA Paper 2010-8066.

[18] F. Gobetz and J. Doll, "A Survey of Impulsive Trajectories, Final Report," Tech. Rep. G-910557-11, United Aircraft Research Laboratories, East Hartford, CT, June 1968.

[19] C. Ocampo and R. Saudemont, "Initial Trajectory Model for a Multi-Maneuver Moon-to-Earth Abort Sequence," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 4, 2010, pp. 1184–1194.

[20] D. Jones and C. Ocampo, "Optimal Impulsive Escape Trajectories from a Circular Orbit to a Hyperbolic Excess Velocity Vector," *AIAA/AAS Astrodynamics Specialist Conference Proceedings*, 2010. AIAA Paper 2010-7524.

[21] L. F. Richardson, "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*, Vol. 210, No. A, 1910, pp. 307–357.

[22] L. F. Richardson and J. A. Gaunt, "The Deferred Approach to the Limit," *Philos. Trans. R. Soc. London*, Vol. 226, No. A, 1927, pp. 299–361.

[23] A. Curtis and J. Reid, "The choice of step lengths when using differences to approximate Jacobian matrices," *IMA Journal of Applied Mathematics*, Vol. 13, No. 1, 1974, pp. 121–126.

[24] R. S. Stepleman and N. D. Winarsky, "Adaptive Numerical Differentiation," *Mathematics of Computation*, Vol. 33, October 1979, pp. 1257–1264.

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

[26] R. Mathur, *An Analytical Approach to Computing Step Sizes for Finite-Difference Derivatives*. PhD thesis, The University of Texas at Austin, Austin, TX, May 2012.

[27] D. Goldberg, "What Every Computer Scientist Should Know about Floating-Point Arithmetic," *ACM Computing Surveys*, Vol. 23, 1991, pp. 5–48.

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

[29] P. Gill, W. Murray, M. Saunders, and M. Wright, "Computing Forward-Difference Intervals for Numerical Optimization," *SIAM Journal of Scientific and Statistical Computing*, Vol. 4, 1983, pp. 310–321.

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

[31] J. Danby, "The Solution of Kepler's Equation," *Celestial Mechanics*, Vol. 40, 1987, pp. 303–312.

[32] R. Restrepo, "GTA: Package Specification," Available on request, October 2011.

[33] W. H. Goodyear, "Completely General Closed-Form Solution for Coordinates and Partial Derivatives of the Two-Body Problem," *Astronomical Journal*, Vol. 70, No. 3, 1965, pp. 189–192.

[34] W. H. Goodyear, "A General Method for the Computation of Cartesian Coordinates and Partial Derivatives of the Two-Body Problem," Tech. Rep. CR-522, NASA, 1966.