49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
4 - 7 January 2011, Orlando, Florida

AIAA 2011-886

The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations

Jeffrey A. Fike* and Juan J. Alonso †
Department of Aeronautics and Astronautics
Stanford University, Stanford, CA 94305

The complex-step approximation for the calculation of derivative information has two significant advantages: the formulation does not suffer from subtractive cancellation errors and it can provide exact derivatives without the need to search for an optimal step size. However, when used for the calculation of second derivatives that may be required for approximation and optimization methods, these advantages vanish. In this work, we develop a novel calculation method that can be used to obtain first (gradient) and second (Hessian) derivatives and that retains all the advantages of the complex-step method (accuracy and step-size independence). In order to accomplish this task, a new number system which we have named *hyper-dual numbers* and the corresponding arithmetic have been developed. The properties of this number system are derived and explored, and the formulation for derivative calculations is presented. Hyper-dual number arithmetic can be applied to arbitrarily complex software and allows the derivative calculations to be free from both truncation and subtractive cancellation errors. A numerical implementation on an unstructured, parallel, unsteady Reynolds-Averaged Navier Stokes (URANS) solver, Joe, is created using C++ operator overloading. Results are presented demonstrating the potential of this method in computations of relevance to the analysis and design of multi-disciplinary aerospace systems.

Nomenclature

f	generic function
f'	first derivative of f
f''	second derivative of f
x, y	generic scalar value; not necessarily real numbers
$\mathbf{x}$	vector of scalar values
n	dimension of $\mathbf{x}$
h, h_1, h_2	step sizes
d	generic step in Taylor series expansion
i, j, k	imaginary terms
$E, E_1, E_2, (E_1E_2)$	generalized-complex terms
$\epsilon, \epsilon_1, \epsilon_2, \epsilon_1\epsilon_2$	dual number terms
p, q, r, s	generic real values
a, b	generic hyper-dual numbers
a_1, a_2, a_3, a_4	real-valued elements of hyper-dual number a
b_1, b_2, b_3, b_4	real-valued elements of hyper-dual number b
x_1, x_2, x_3, x_4	real-valued elements of hyper-dual number x
$t_0, t_1, ..., t_8$	intermediate stages in a function evaluation procedure
C_L	aerodynamic lift coefficient
C_D	aerodynamic drag coefficient
C_M	aerodynamic pitching moment coefficient

*Graduate Student, AIAA Student Member.
†Associate Professor, AIAA Senior Member.

M	free stream Mach number
α, AOA	free stream angle of attack
γ	ratio of specific heats
θ	wedge angle
β	oblique shock angle
M_1	Mach number before a shock
P_1	pressure before a shock
P_2	pressure after a shock
ψ_1, ψ_2, ψ_3	Lagrange multipliers in adjoint formulation

I. Introduction

First-derivative information is used in many methodologies that are routinely employed in engineering, including numerical optimization, response surface / surrogate model creation, uncertainty quantification, and mesh refinement / adaptation. In many of these applications, second-derivative information can also be useful if it could be obtained both accurately and efficiently.

The first-derivative complex-step approximation[1] has several advantages when compared to alternative first-derivative calculation methods. For instance, finite-difference formulas are easy to implement, but they are subject to both truncation and subtractive cancellation errors. The accuracy of finite-difference derivatives depends on the choice of step size, which involves a tradeoff between the two sources of error. The first-derivative complex-step approximation is subject to truncation error, but it is immune to subtractive cancellation error. This allows the first-derivative complex-step approximation to be made effectively exact, to numerical precision, by choosing a very small step size. This increase in accuracy comes at the expense of a slight increase in the difficulty of implementation and in the cost of computation. The increase in difficulty is related to the conversion of an analysis code that operates on real numbers to one that operates on complex numbers instead. This added difficulty also includes the necessary re-definition of all the operations that involve complex numbers. While this may require more than simply changing the variable type declarations in the programs, it has been shown that the procedure can be fully automated for codes of arbitrary complexity[2] written in MATLAB, Fortran 90/95, C, and C++ (among others).

The first-derivative complex-step approximation is, by no means, as computationally efficient as an adjoint method[3] for computing all of the first derivatives of a function of many variables. However, adjoint methods are, comparatively, much more difficult to implement and, under many circumstances, the creation of a robust adjoint capability still remains a research task. Automatic differentiation techniques[4,5] can also be used to produce derivatives using the forward and reverse modes, but can lead to significant memory and computational requirements if the differentiation procedures are applied to entire simulation codes and not only some portions of them. In many situations, the computational cost associated with the complex-step approximation can be prohibitive, and an adjoint method is preferred. However, even in such situations, the first-derivative complex-step approximation can still be used to aid in the development and verification of the sensitivities produced by the adjoint code.[3]

In many applications of engineering interest (optimization, approximation theory / surrogate model development, uncertainty quantification, etc.) the curvature of the problem must be identified either by direct calculation of the Hessian or through multiple evaluations of the gradient vector at different points in the design space. In many of these situations, it would be beneficial to be able to compute second derivatives accurately and robustly, yet not many methods have been developed for this purpose. Some automatic differentiation techniques offer the possibility of computing Hessian information[5] but the same shortcomings mentioned above (for first derivatives) exist. Finite differences can be used to compute the second derivatives in the Hessian, but the problems with subtractive cancellation and optimal step size selection (in two computational directions) are compounded.

A second-derivative calculation method is sought that possesses the same advantages of the complex-step method for first derivatives. It should provide increased accuracy with only a slight increase in the difficulty of the implementation (when compared to finite-differences). It should also be relatively easy to implement, so that it is useful when, for example, developing second-order adjoint methods. The desired second-derivative calculation method should therefore be immune to subtractive cancellation error and should be automatically implementable in arbitrarily-complex analysis programs, as was the case with the complex-

step approximation. This paper describes the development of such a method based on hyper-dual numbers.

It will be shown that second-derivative complex-step approximations are subject to subtractive cancellation error and therefore not the desired method. An examination of this behavior suggests that a number with multiple non-real parts might produce the desired method. Quaternions are found to be unsuitable because multiplication is not commutative. The properties of a number system with three non-real parts, where multiplication is commutative are discussed. Out of the many possible number systems, hyper-dual numbers are chosen to produce the most accurate derivative calculations.

Hyper-dual numbers are a higher-dimensional extension of dual numbers in the same way that quaternions are a higher-dimensional extension of ordinary complex numbers. Dual numbers and ordinary complex numbers are types of generalized-complex numbers.[6] Dual numbers are based on the non-real term $\epsilon^2 = 0$ where $\epsilon \neq 0$,[7,8] instead of $i^2 = -1$, which is used for complex numbers. The resulting hyper-dual number calculation method is not subject to either truncation error or subtractive cancellation error and, therefore, the results produced are exact.

In this paper, we first present more details of the motivation for the development of these new hyper-dual numbers and discuss in detail the shortcomings of the complex-step method for second-derivative calculation for which, admittedly, this method was not developed. We then discuss the mathematical properties and the definition of functions of hyper-dual numbers and present the details of the implementation. For our numerical implementation of hyper-dual numbers we have chosen to describe them as a class, using operator overloading, in programming languages such as C++ and MATLAB. We finally present a comparison of the various first- and second-derivative calculation methods for a simple, analytic function, demonstrating the accuracy of the hyper-dual number calculations. The same implementation is used to create a hyper-dual number version of a parallel, unstructured, three-dimensional unsteady Reynolds-Averaged Navier-Stokes computational fluid dynamics (CFD) code, Joe.[9] The hyper-dual number version of this CFD code is used to compute the first and second derivatives of the aerodynamic coefficients for a turbulent, transonic airfoil, which are compared with finite-difference results. The inviscid, supersonic flow over a wedge is also analyzed, and the hyper-dual number calculations are compared to exact analytic results.

II. Motivation

A second-derivative calculation method is sought that has comparable performance, based on accuracy, computational efficiency, and ease of implementation, to the first-derivative complex-step approximation. The primary drivers for creation of this method will be accuracy, ease of implementation, and insensitivity to choices of step size. The computational efficiency of the resulting method will be discussed in a later section.

A. First-Derivatives

Before developing a second-derivative method, it is instructive to compare the first-derivative complex-step and finite-difference approximations. Two common finite-difference approximations are the forward-difference approximation

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

and the central-difference approximation

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

These finite-difference approximations can be derived using the Taylor series

$$f(x+d) = f(x) + df'(x) + \frac{1}{2!}d^2 f''(x) + \frac{d^3 f'''(x)}{3!} + \dots, \tag{3}$$

for real steps $d = \pm h$. The first-derivative complex-step approximation[1] is also derived from the Taylor series, Eq. (3), except that an imaginary step, $d = ih$, is used instead of a real-valued one. This results in

$$f(x+ih) = f(x) + ihf'(x) - \frac{1}{2!}h^2 f''(x) - i\frac{h^3 f'''(x)}{3!} + \dots. \tag{4}$$

As with any complex number this can be separated into real and imaginary parts

$$f(x+ih) = \left(f(x) - \frac{1}{2!}h^2 f''(x) + ...\right) + ih\left(f'(x) - \frac{1}{3!}h^2 f'''(x) + ...\right). \tag{5}$$

The first derivative is the leading term of the imaginary part of this expression. This allows an approximation for the first derivative to be formed simply by taking the imaginary part and dividing by the step size,

$$f'(x) = \frac{Imag[f(x+ih)]}{h} + O(h^2). \tag{6}$$

There are two main sources of error that affect these derivative approximations: truncation error and subtractive cancellation error. Truncation error is a fundamental part of the derivative approximation formula and is related to the higher-order terms that are neglected. For instance, the truncation error of Eq. (6) is $O(h^2)$, and is due to the third- and higher-order terms that are being ignored. The truncation error can be minimized by making the step size, h, very small. Equations (1) and (2) are also subject to subtractive cancellation error, due to the appearance of difference operations (between numbers that are very close to each other) in the formulas. Subtractive cancellation error only occurs because in digital computers, numbers are represented with a finite number of bits: as the numbers being differenced become so close to each other that they cannot be told apart in a digital computer (this occurs when the numbers are the same to 14-16 significant digits in double-precision arithmetic), the difference is effectively zero, leading to an incorrect approximation to the value of the derivative. Since subtractive cancellation error becomes more significant when the step size becomes small, we can quickly understand the challenge: we require small step sizes to minimize truncation error but that leads to subtractive cancellation errors. With finite-difference approximations, we cannot avoid truncation and subtractive cancellation errors simultaneously.

In summary, for accurate finite-difference derivative approximations, the step size needs to be small enough to reduce the truncation error below a specified threshold, but not so small that the subtractive cancellation error begins to dominate. There are several difficulties that arise when trying to find the optimal step size. Typically, this optimal step size is problem dependent. It may also vary depending on the value of the independent variable for which the derivative is being computed. More importantly, for functions of multiple variables, it may be different for each variable. As shown in Eq. (6), the first-derivative complex-step approximation does not involve a difference operation and therefore does not suffer from subtractive cancellation error. Figure 1(a) shows the behavior of the complex-step and finite-difference approximations for first derivatives, as a function of step size. As shown, the error of the first-derivative complex-step approximation can be reduced to numerical accuracy by making the step size very small.

B. Second Derivatives

The above discussion has focused on first-derivative calculations. Finite-difference formulas for second derivatives have essentially the same behavior as those for first derivatives. However, the range of acceptable step sizes is even narrower, making the appropriate choice of h even more important. The behavior of these methods is shown in Fig. 1(b). Unfortunately, the optimal step size for accurate second derivatives is usually not the same as the optimal step size for first derivatives. A single step size can be chosen as a compromise, or separate step sizes can be used for the first- and second-derivative calculations, requiring additional function evaluations.

A second-derivative complex-step approximation can be derived from Eq. (5),

$$f''(x) = \frac{2\left(f(x) - Real[f(x+ih)]\right)}{h^2} + O(h^2). \tag{7}$$

This formula involves a difference operation and is therefore subject to subtractive cancellation error, this behavior is shown in Fig. 1(b). It is possible to create alternative approximations that use multiple, different complex-steps,[10] but while these alternative formulations may offer improvements over Eq. (7), they still suffer from subtractive cancellation error. As with the finite-difference formulas, there may be a conflict between the optimal step size for accurate first derivatives and that for accurate second derivatives.

For first derivatives the complex-step approximation was immune to subtractive cancellation error because the first derivative was the leading term of the imaginary part. In order for a second-derivative approximation to be immune to subtractive cancellation error, the second-derivative term should also be the leading term

of an *imaginary* part. Second derivatives are not often of interest by themselves, but are typically used along with first derivatives. This implies that, for a better method to exist, the first and second derivatives should both be leading terms of an expansion. It is impossible to form such a second-derivative approximation using complex numbers, which only have one non-real part. This suggests that the new method that we are seeking *should use a number with multiple non-real parts.*

Another issue with second-derivative complex-step approximations is the method for computing the cross-derivatives for a function of multiple variables. An approximation for the cross-derivatives can be formed from the Taylor series for a function of two variables, where an imaginary step, ih, is applied to each variable. This results in the approximation formula

$$\frac{\partial^2 f(x,y)}{\partial x \partial y} \approx \frac{1}{2}\left(\frac{2\left(f(x,y) - Real\left[f(x+ih, y+ih)\right]\right)}{h^2} - \frac{\partial^2 f(x,y)}{\partial x^2} - \frac{\partial^2 f(x,y)}{\partial y^2}\right). \tag{8}$$

This approximation relies on the computation of the pure-derivative terms, resulting in a sequential calculation where the error is cumulative. The cross-derivative approximation has this behavior because the perturbation applied to each variable uses the same non-real value, i. Ideally, the perturbation of each variable would use different non-real values, again suggesting the use of a number with *multiple non-real parts.*

III. Derivation

The above discussion suggests that a number with multiple non-real parts should be used to avoid subtractive cancellation error and allow for the independent calculation of the cross-derivative terms. This, in turn, suggests that a logical first attempt at creating a new approximation method is to use quaternions, which have three imaginary components, instead of complex numbers. The imaginary numbers used by quaternions have the properties $i^2 = j^2 = k^2 = -1$ and $ijk = -1$.

An approximation can be formed by looking at the Taylor series, Eq. (3), with a quaternion step $d = h_1 i + h_2 j + 0k$. The properties of this approximation can be examined by looking at the powers of d,

$$d^2 = -\left(h_1^2 + h_2^2\right), \tag{9}$$

$$d^3 = -\left(h_1^2 + h_2^2\right)\left(h_1 i + h_2 j + 0k\right), \tag{10}$$

$$d^4 = \left(h_1^2 + h_2^2\right)^2, \ldots . \tag{11}$$

Ideally, the second-derivative term would be the leading term of the k part. Instead, the k part is always zero and the second-derivative term is only part of the real component of $f(x + h_1 i + h_2 j + 0k)$. An approximation formula for the second derivative can be formed,

$$f''(x) = \frac{2\left(f(x) - Real[f(x + h_1 i + h_2 j + 0k)]\right)}{h_1^2 + h_2^2} + O(h_1^2 + h_2^2), \tag{12}$$

but this approximation is also subject to subtractive cancellation error. The problem with using quaternions is that multiplication is not commutative, $ij = k$ but $ji = -k$. This puts the second-derivative term in the real part, requiring a difference operation to approximate the second derivative and extract the required information.

A different number similar in form to the quaternions, with three non-real components, might produce the desired approximation if multiplication in the associated arithmetic were commutative. This is not possible using the standard definition of imaginary numbers defined as the square root of -1, but it might be possible with some other definition. Consider a number with three non-real components E_1, E_2, $(E_1 E_2)$ where $E_1 E_2 = E_2 E_1$. The values of E_1^2, E_2^2, and $(E_1 E_2)^2$ can be determined by examining the values of d and its powers from the Taylor series in Eq. (3),

$$d = h_1 E_1 + h_2 E_2 + 0 E_1 E_2, \tag{13}$$

$$d^2 = h_1^2 E_1^2 + h_2^2 E_2^2 + 2 h_1 h_2 E_1 E_2, \tag{14}$$

$$d^3 = h_1^3 E_1^3 + 3 h_1 h_2^2 E_1 E_2^2 + 3 h_1^2 h_2 E_1^2 E_2 + h_2^3 E_2^3, \tag{15}$$

$$d^4 = h_1^4 E_1^4 + 6 h_1^2 h_2^2 E_1^2 E_2^2 + 4 h_1^3 h_2 E_1^3 E_2 + 4 h_1 h_2^3 E_1 E_2^3 + h_2^4 E_2^4. \tag{16}$$

The first term with a non-zero (E_1E_2) component is d^2, which means that the second derivative is the leading term of the (E_1E_2) component. This means that second-derivative approximations can be formed that are not subject to subtractive cancellation error. This is true as long as $E_1E_2 \neq 0$, regardless of the values of E_1^2, E_2^2, and $(E_1E_2)^2$. The only restriction is that multiplication must be commutative, i.e. $E_1E_2 = E_2E_1$.

It must be noted that the values of E_1^2, E_2^2, and $(E_1E_2)^2$ are not completely independent. The requirement that $E_1E_2 = E_2E_1$ produces the constraint

$$(E_1E_2)^2 = E_1E_2E_1E_2 = E_1E_1E_2E_2 = E_1^2E_2^2. \tag{17}$$

Even satisfying this constraint still leaves many possibilities regarding the definition of the numbers we are seeking. One possibility is to use $E_1^2 = E_2^2 = -1$ which results in $(E_1E_2)^2 = 1$. These are known as circular fourcomplex numbers.[11] Another approach is to constrain $E_1^2 = E_2^2 = (E_1E_2)^2$. This leads to two possibilities $E_1^2 = E_2^2 = (E_1E_2)^2 = 0$ and $E_1^2 = E_2^2 = (E_1E_2)^2 = 1$. When looking at these terms it is important to note that $E_1 \neq 0$ when $E_1^2 = 0$. The terms E_1, E_2, and (E_1E_2) are not real numbers, so it is possible to have $E_1 \neq 0$ when $E_1^2 = 0$. These terms can be thought of in a similar manner to imaginary numbers, as being directions off the real number line.

This interpretation is consistent with generalized-complex numbers. Generalized-complex numbers are numbers of the form $p+qE$. Addition for these numbers is defined as $(p+qE)+(r+sE) = (p+r)+(q+s)E$, and multiplication is defined as $(p+qE)(r+sE) = pr + (ps+qr)E + qsE^2$. There are three types of generalized-complex numbers, each based on a different definition of E^2. Ordinary complex numbers are based on $E^2 = -1$. The other two possibilities are dual numbers based on $E^2 = 0$ and double numbers based on $E^2 = 1$.

By examining Eqs. (13) thru (16), the best derivative approximation is formed by taking $E_1^2 = E_2^2 = (E_1E_2)^2 = 0$. To distinguish this situation from other definitions of E_1^2, E_2^2, and$(E_1E_2)^2$, and to emphasize the connection to dual numbers which use $\epsilon^2 = 0$, in the notation for these hyper-dual numbers we will use ϵ instead of E, and we will require that $\epsilon_1^2 = \epsilon_2^2 = (\epsilon_1\epsilon_2)^2 = 0$. Using this definition, d^3 and all higher powers of d are identically zero. The Taylor series truncates *exactly* at the second-derivative term, yielding the properties of the approximation that we are seeking:

$$f(x + h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2) = f(x) + h_1f'(x)\epsilon_1 + h_2f'(x)\epsilon_2 + h_1h_2f"(x)\epsilon_1\epsilon_2. \tag{18}$$

In this equation, there is *no truncation error*, a seemingly implausible occurrence that is only possible due to the use of hyper-dual numbers and a different arithmetic. The first and second derivatives are now the leading terms of the non-real parts, meaning that these values can simply be found by examining the non-real parts of the number, and therefore the derivative calculations are not subject to subtractive cancellation errors. Therefore, the use of hyper-dual numbers results in first- and second-derivative calculations that are exact, regardless of the step size. The real part is also exactly the same as the function evaluated for a real number, x.

In the complex-step approximation for first derivatives, the value of the derivative is found by taking the imaginary part of $f(x+ih)$ and dividing by h. Similarly, first derivative information is obtained from a hyper-dual function evaluation by taking the ϵ_1 and ϵ_2 parts of $f(x+h_1\epsilon_1+h_2\epsilon_2+0\epsilon_1\epsilon_2)$ and dividing by h_1 and h_2, respectively. The value of the second derivative is obtained by taking the $\epsilon_1\epsilon_2$ part of $f(x+h_1\epsilon_1+h_2\epsilon_2+0\epsilon_1\epsilon_2)$ and dividing by h_1h_2.

For functions of multiple variables, $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$, first derivatives are found using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{\epsilon_1 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_1} \tag{19}$$

and

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = \frac{\epsilon_2 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_2}. \tag{20}$$

And second derivatives are found using

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \frac{\epsilon_1\epsilon_2 part\left[f(\mathbf{x} + h_1\epsilon_1\mathbf{e}_i + h_2\epsilon_2\mathbf{e}_j + \mathbf{0}\epsilon_1\epsilon_2)\right]}{h_1h_2}, \tag{21}$$

where $\mathbf{e}_i$ and $\mathbf{e}_j$ are unit vectors composed of all zeros except the i^{th} and j^{th} components, respectively.

The formulas in Eqs. (19) thru (21) represent the outcome we were seeking: expressions for first and second derivatives that are exact, are not subject to subtractive cancellation error, and that can be computed easily in existing computational analysis codes. These formulas are made possible by the understanding and development of hyper-dual numbers with the properties that have been described above.

IV. Implementation

A. Mathematical Properties

Before a function like $f(x + h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2)$ can be evaluated, the mathematics of hyper-dual numbers need to be developed. This includes defining basic operations like addition, multiplication and division, and comparison operations like $>$ or $==$, as well as how trigonometric and transcendental functions such as sine, logarithms, and the absolute value behave.

The rules for adding and multiplying two hyper-dual numbers are straightforward. Given two hyper-dual numbers $a = a_1 + a_2\epsilon_1 + a_3\epsilon_2 + a_4\epsilon_1\epsilon_2$ and $b = b_1 + b_2\epsilon_1 + b_3\epsilon_2 + b_4\epsilon_1\epsilon_2$, addition is defined as

$$a + b = (a_1 + b_1) + (a_2 + b_2)\epsilon_1 + (a_3 + b_3)\epsilon_2 + (a_4 + b_4)\epsilon_1\epsilon_2, \tag{22}$$

and multiplication is defined as

$$ab = (a_1b_1) + (a_1b_2 + a_2b_1)\epsilon_1 + (a_1b_3 + a_3b_1)\epsilon_2 + (a_1b_4 + a_2b_3 + a_3b_2 + a_4b_1)\epsilon_1\epsilon_2. \tag{23}$$

From these definitions, other mathematical operations can be defined, such as the multiplicative inverse

$$a^{-1} = \frac{1}{a} = \frac{1}{a_1} - \frac{a_2}{a_1^2}\epsilon_1 - \frac{a_3}{a_1^2}\epsilon_2 + \left(-\frac{a_4}{a_1^2} + \frac{2a_2a_3}{a_1^3}\right)\epsilon_1\epsilon_2. \tag{24}$$

As can be seen from this expression, division, $\frac{b}{a} = ba^{-1}$, is limited to numbers, a, with a non-zero real part, a_1. One of the requirements, for a number system to be a division algebra, is that a multiplicative inverse exists for every number that is not equal to zero. In other words, every number except $0 + 0\epsilon_1 + 0\epsilon_2 + 0\epsilon_1\epsilon_2$ should have an inverse. Instead, a hyper-dual number has an inverse for every number except $0 + a_2\epsilon_1 + a_3\epsilon_2 + a_4\epsilon_1\epsilon_2$. However, in practice this should not be an issue. The application of hyper-dual numbers that is being considered here is converting a function or analysis code, which operates on real numbers, to operate on hyper-dual numbers. Division by zero should never occur in the real valued function, so division by hyper-dual numbers with real parts, $a_1 = 0$, should also never occur.

This discussion suggests that the definition for the norm of a hyper-dual number should be based on the real part of the number only, $norm(a) = \sqrt{a_1^2}$. This definition of the norm has the property that $norm(a * b) = norm(a) * norm(b)$. Defining the norm of a hyper-dual number to only involve the real part of the number suggests that comparisons between two hyper-dual numbers, or between a hyper-dual number and a real number, should also only be based on the real part of the number. In other words, $a < b$ is equivalent to $a_1 < b_1$ and $a == b$ is equivalent to $a_1 == b_1$. This means that any comparisons performed in the hyper-dual analysis code will produce the same results as in the real analysis code. This allows a code with branching statements to follow the same execution path when operating on both real numbers and hyper-dual numbers.

The behavior of functions that operate on hyper-dual numbers also needs to be developed. This can be accomplished by considering a function, $f(x)$, with $x = x_1 + x_2\epsilon_1 + x_3\epsilon_2 + x_4\epsilon_1\epsilon_2$. The Taylor series expansion of this function is

$$f(x) = f(x_1) + x_2f'(x_1)\epsilon_1 + x_3f'(x_1)\epsilon_2 + (x_4f'(x_1) + x_2x_3f''(x_1))\,\epsilon_1\epsilon_2. \tag{25}$$

Therefore a function, like $sin(x)$, can be written as

$$sin(x) = sin(x_1) + x_2cos(x_1)\epsilon_1 + x_3cos(x_1)\epsilon_2 + (x_4cos(x_1) - x_2x_3sin(x_1))\,\epsilon_1\epsilon_2. \tag{26}$$

Other functions, such as max(a,b) or the absolute value of x, $|x|$, can be defined by treating the function as a procedure. For instance, the absolute value of $x = x_1 + x_2\epsilon_1 + x_3\epsilon_2 + x_4\epsilon_1\epsilon_2$ can be computed simply as x, or $-x$ if $x < 0$, i.e. $x_1 < 0$. One other slight complication is that some functions may have infinite valued first or second derivatives, such as $sqrt(x)$ at $x = 0$. To avoid any numerical issues associated with infinite values, the hyper-dual versions of these functions are defined to return a very large, but finite, value for the derivatives.

B. Software Implementation

The discussion so far has dealt primarily with issues that affect the accuracy of this second-derivative calculation method, but another concern is the ease of implementation. To use the complex-step approximation, a real valued code needs to be converted to use complex numbers. This basically only requires changing the variable type declarations, and including the appropriate libraries if necessary, so that the computer knows how to perform the various mathematical operations on complex numbers. The computer program needs to know two things: firstly how to store numbers of a given type, and secondly how to manipulate these numbers in a manner that is consistent with the arithmetic of the numbers being used (either complex or hyper-dual numbers).

There are several approaches one could use to accomplish this. One approach to allocating hyper-dual numbers is to redefine every real scalar variable as a four component real-valued vector. Every function could then be changed to operate on these vectors instead of the real-valued scalars. This would involve replacing statements like $a = b + c$ with a function like $a = HDadd(b, c)$, which would make converting a code to use hyper-dual numbers very difficult. An alternative approach is to define a hyper-dual number class, which uses operator overloading. Operator overloading enables the redefinition of operators such as +, *, >, etc. for user-defined data types. The use of operator overloading allows the majority of the real valued code to be left unmodified. Using this approach means that converting a real valued code to use hyper-dual numbers involves little more than changing the variable types, which is the same amount of work required to use the complex-step approximation.

In some situations there is a little more work that needs to be done. Codes that use MPI will need to define a hyper-dual MPI data type and operations. Reading or writing to a file or the standard output may also need to be modified, as will any calls to external packages, such as PETSc. C++, MATLAB, and MPI implementations have been created and are available from our website.[12] These implementations are by no means exhaustive; only those functions that have been needed have been implemented so far, but extensions should be straightforward.

C. Variations

Following the above discussion, methods for computing exact higher derivatives can be created by using more non-real parts. For instance, exact third-derivatives can be computed by including a third non-real part ϵ_3. The coefficients of the Taylor series, Eq. (3), then become

$$\begin{aligned} d &= h_1\epsilon_1 + h_2\epsilon_2 + h_3\epsilon_3 + 0\epsilon_1\epsilon_2 + 0\epsilon_1\epsilon_3 + 0\epsilon_2\epsilon_3 + 0\epsilon_1\epsilon_2\epsilon_3, && (27)\\ d^2 &= 2h_1h_2\epsilon_1\epsilon_2 + 2h_1h_3\epsilon_1\epsilon_3 + 2h_2h_3\epsilon_2\epsilon_3, && (28)\\ d^3 &= 6h_1h_2h_3\epsilon_1\epsilon_2\epsilon_3, && (29)\\ d^4 &= 0, && (30) \end{aligned}$$

which results in an exact third-derivative calculation.

Exact first derivatives can be found by using dual numbers, which have only one non-real component, ϵ. An implementation of dual numbers is also available.[12] There is little benefit to using dual numbers to compute first derivatives compared to using complex numbers. Both methods require the same amount of effort to use, and can produce results with the same level of accuracy, to numerical precision. Manipulating dual numbers does require fewer mathematical operations than manipulating ordinary complex numbers. But the efficiency of the standard complex number implementation may reduce this benefit.

The use of hyper-dual numbers can also be extended to compute derivatives of complex-valued functions. A partial implementation and example program are available[12] demonstrating this on a complex version of Eq. (31) and yielding a technique that streamlines other modifications of the complex-step method.[13]

V. Results

A. Analytic Test Function

The behavior of the first- and second-derivative calculation methods, discussed above, is demonstrated using a simple, analytic function,[1]

$$f(x) = \frac{e^x}{\sqrt{sin(x)^3 + cos(x)^3}}. \quad (31)$$

The exact first derivative of this function is

$$f'(x) = \frac{e^x \left(3\cos x + 5\cos 3x + 9\sin x + \sin 3x\right)}{8\left(\sin^3 x + \cos^3 x\right)^{\frac{3}{2}}}, \tag{32}$$

and the second derivative is given by

$$f''(x) = \frac{e^x \left(130 - 12\cos 2x + 30\cos 4x + 12\cos 6x - 111\sin 2x + 48\sin 4x + 5\sin 6x\right)}{64\left(\sin^3 x + \cos^3 x\right)^{\frac{5}{2}}}. \tag{33}$$

The function values and derivatives are computed numerically at $x = 1.5$.

Equation (31) can be computed using the following procedure:

$$\begin{aligned}
t_0 &= x \\
t_1 &= e^{t_0} &&= e^x \\
t_2 &= \sin t_0 &&= \sin x \\
t_3 &= t_2^3 &&= \sin^3 x \\
t_4 &= \cos t_0 &&= \cos x \\
t_5 &= t_4^3 &&= \cos^3 x \\
t_6 &= t_3 + t_5 &&= \left(\sin^3 x + \cos^3 x\right) \\
t_7 &= t_6^{-0.5} &&= \left(\sin^3 x + \cos^3 x\right)^{-\frac{1}{2}} \\
t_8 &= t_1 * t_7 &&= \frac{e^x}{\sqrt{\sin^3 x + \cos^3 x}}.
\end{aligned}$$

When using hyper-dual numbers, this procedure produces the following results. The symbolic results were manipulated by hand and given to show that the derivative results are exact. The numerical results show the calculations that are actually performed when hyper-dual numbers are used in C++ or MATLAB.

$$\begin{aligned}
t_0 &= x \\
&= x + h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2 \\
&= 1.5 + 1.0\epsilon_1 + 1.0\epsilon_2 + 0.0\epsilon_1\epsilon_2 \\
t_1 &= e^{t_0} \\
&= e^x + h_1 e^x\epsilon_1 + h_2 e^x\epsilon_2 + h_1 h_2 e^x\epsilon_1\epsilon_2 \\
&= 4.4817 + 4.4817\epsilon_1 + 4.4817\epsilon_2 + 4.4817\epsilon_1\epsilon_2 \\
t_2 &= \sin t_0 \\
&= \sin x + h_1\cos x\epsilon_1 + h_2\cos x\epsilon_2 - h_1 h_2\sin x\epsilon_1\epsilon_2 \\
&= 0.99749 + 0.070737\epsilon_1 + 0.070737\epsilon_2 - 0.99749\epsilon_1\epsilon_2 \\
t_3 &= t_2^3 \\
&= \sin^3 x + 3h_1\cos x\sin^2 x\epsilon_1 + 3h_2\cos x\sin^2 x\epsilon_2 - \frac{3}{4}h_1 h_2\left(\sin x - 3\sin 3x\right)\epsilon_1\epsilon_2 \\
&= 0.9925 + 0.21115\epsilon_1 + 0.21115\epsilon_2 - 2.9476\epsilon_1\epsilon_2 \\
t_4 &= \cos t_0 \\
&= \cos x - h_1\sin x\epsilon_1 - h_2\sin x\epsilon_2 - h_1 h_2\cos x\epsilon_1\epsilon_2 \\
&= 0.070737 - 0.99749\epsilon_1 - 0.99749\epsilon_2 - 0.070737\epsilon_1\epsilon_2 \\
t_5 &= t_4^3 \\
&= \cos^3 x - 3h_1\cos^2 x\sin x\epsilon_1 - 3h_2\cos^2 x\sin x\epsilon_2 - \frac{3}{4}h_1 h_2\left(\cos x + 3\cos 3x\right)\epsilon_1\epsilon_2 \\
&= 0.00035395 - 0.014974\epsilon_1 - 0.014974\epsilon_2 + 0.42124\epsilon_1\epsilon_2 \\
t_6 &= t_3 + t_5 \\
&= \left(\sin^3 x + \cos^3 x\right) + 3h_1\cos x\sin x\left(\sin x - \cos x\right)\epsilon_1 + 3h_2\cos x\sin x\left(\sin x - \cos x\right)\epsilon_2
\end{aligned}$$

$$
\begin{aligned}
& -\frac{3}{4}h_1h_2\left(\sin x+\cos x+3\cos 3x-3\sin 3x\right)\epsilon_1\epsilon_2 \\
= \;& 0.99286+0.19618\epsilon_1+0.19618\epsilon_2-2.5263\epsilon_1\epsilon_2 \\
t_7 \;=\; & t_6^{-0.5} \\
= \;& \left(\sin^3 x+\cos^3 x\right)^{-\frac{1}{2}}-\frac{3}{2}h_1\frac{\cos x\sin x\left(\sin x-\cos x\right)}{\left(\sin^3 x+\cos^3 x\right)^{\frac{3}{2}}}\epsilon_1-\frac{3}{2}h_2\frac{\cos x\sin x\left(\sin x-\cos x\right)}{\left(\sin^3 x+\cos^3 x\right)^{\frac{3}{2}}}\epsilon_2 \\
& +\frac{3}{64}h_1h_2\frac{\left(30+2\cos 4x-41\sin 2x+3\sin 6x\right)}{\left(\sin^3 x+\cos^3 x\right)^{\frac{5}{2}}}\epsilon_1\epsilon_2 \\
= \;& 1.0036-0.099148\epsilon_1-0.099148\epsilon_2+1.3062\epsilon_1\epsilon_2 \\
t_8 \;=\; & t_1*t_7 \\
= \;& \frac{e^x}{\sqrt{\sin^3 x+\cos^3 x}}+h_1\frac{e^x\left(3\cos x+5\cos 3x+9\sin x+\sin 3x\right)}{8\left(\sin^3 x+\cos^3 x\right)^{\frac{3}{2}}}\epsilon_1 \\
& +h_2\frac{e^x\left(3\cos x+5\cos 3x+9\sin x+\sin 3x\right)}{8\left(\sin^3 x+\cos^3 x\right)^{\frac{3}{2}}}\epsilon_2 \\
& +h_1h_2\frac{e^x\left(130-12\cos 2x+30\cos 4x+12\cos 6x-111\sin 2x+48\sin 4x+5\sin 6x\right)}{64\left(\sin^3 x+\cos^3 x\right)^{\frac{5}{2}}}\epsilon_1\epsilon_2 \\
= \;& 4.4978+4.0534\epsilon_1+4.0534\epsilon_2+9.4631\epsilon_1\epsilon_2
\end{aligned}
$$

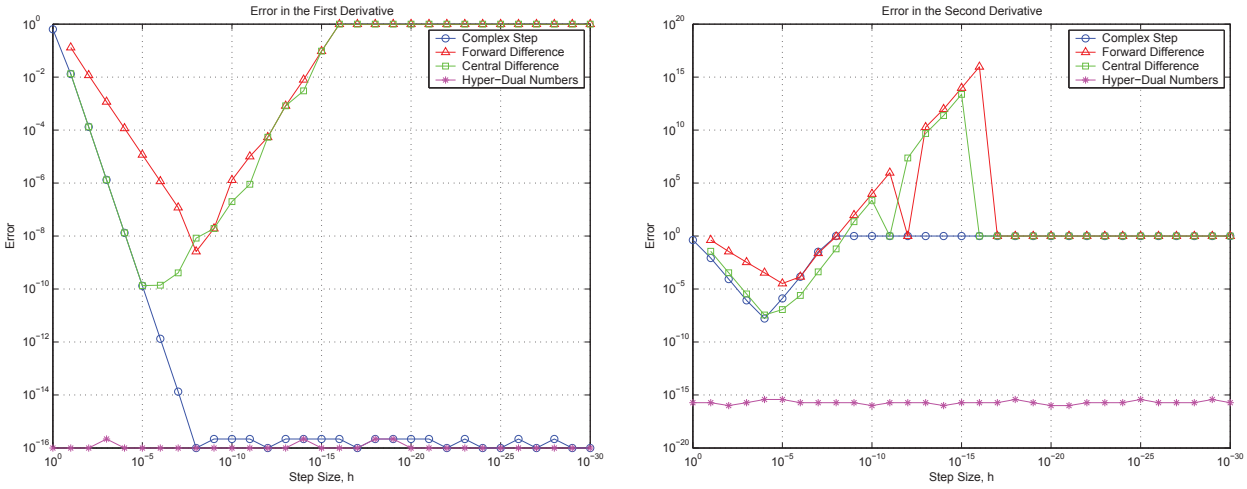
The real component of the symbolic evaluation of t_8 in the above procedure is exactly the function given in Eq. (31). The ϵ_1 and ϵ_2 components of t_8, when divided by h_1 and h_2 respectively, are exactly the first derivatives as given in Eq. (32). Finally, the $\epsilon_1\epsilon_2$ component is exactly the second derivative, as given in Eq. (33), when divided by h_1h_2. Furthermore, at each stage of the procedure, the real component is exactly the same as in the corresponding real-valued procedure, the ϵ_1 and ϵ_2 components at each stage contain the first derivative of the real component, and the $\epsilon_1\epsilon_2$ component contains the second derivative of the real component. In some senses this resembles the connection between the complex-step and the forward mode of algorithmic differentiation that was presented in Ref. 14.

Figure 1(a) shows the error of several first-derivative calculation methods as function of step size, $h = h_1 = h_2$. As the step size is initially decreased, the error decreases according to the order of the truncation error of the method. However, after a certain point, the error for the finite-difference approximations begins to grow, while the error for the complex-step approximation continues to decrease until it reaches (and remains at) machine zero. This illustrates the effect of subtractive cancellation errors, which affects the finite-difference approximations but not the first-derivative complex-step approximation. The error of the hyper-dual number calculations, which are not subject to truncation error or subtractive cancellation error, is machine zero regardless of step size.

The data shown in the plot has been manipulated slightly to produce a better-looking graph. The error calculated for some of the step sizes was actually reported as zero. This cannot be plotted on a logarithmic graph, so there were gaps in the data when plotted. To resolve this, any errors equal to zero were replaced by an error of 10^{-16}.

Figure 1(b) shows the error of the second-derivative calculation methods as function of step size, h. Again, as the step size is initially decreased, the error of the finite-difference and complex-step approximations behaves according to order of the truncation error. For second derivatives, the complex-step approximation is subject to subtractive cancellation error, as are the finite-difference approximations. The subtractive cancellation error begins to dominate the overall error as the step size is reduced below 10^{-4} or 10^{-5}. Again, the error of the hyper-dual number calculations is machine zero for any step size.

Comparing the behavior of the first- and second-derivative approximations, Figs. 1(a) and 1(b) show the difficulty of computing accurate second derivatives using traditional methods. The first-derivative approximations allow for a wider range of acceptable step sizes, and the error of the approximations is roughly two orders of magnitude lower than for the second-derivative approximations.



(a) First-Derivative Accuracy.

(b) Second-Derivative Accuracy.

Figure 1. The accuracies of several derivative calculation methods are presented as a function of step size.

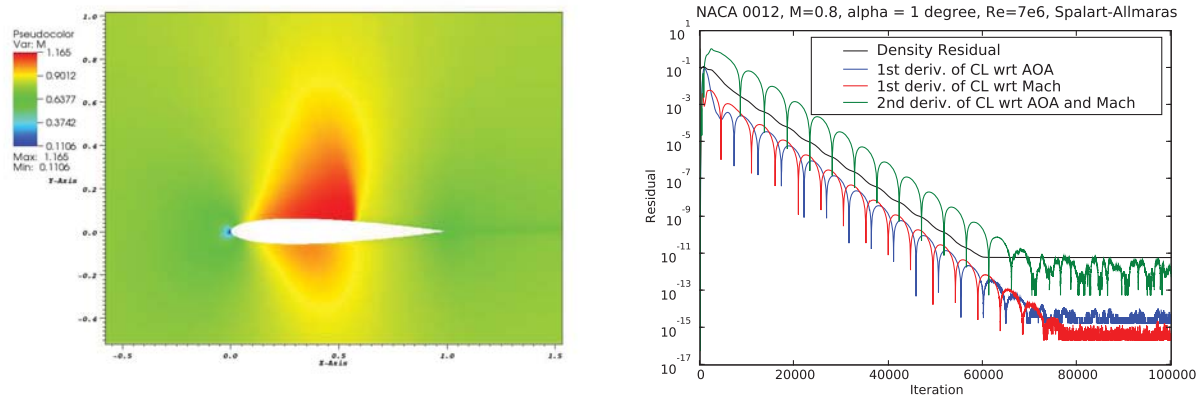
B. CFD Results

The hyper-dual number implementation described above has also been used to produce first and second derivatives of quantities computed using a CFD code. The CFD code used is Joe,[9] a parallel, unstructured, 3-D, unsteady Reynolds-averaged Navier-Stokes code developed under the sponsorship of the Department of Energy under the Predictive Science Academic Alliance Program. This code is written in C++, which enables the straightforward conversion to hyper-dual numbers. The conversion of a real-valued analysis code involves modifying or overloading each function to operate on hyper-dual numbers. This requires modifications to the source code. However, CFD codes, such as Joe, often make use of external libraries for which the source code is not available. For instance Joe can make use of the preconditioned generalized minimum residual method (GMRES) from the Portable, Extensible Toolkit for Scientific Computation (PETSc). We can still make use of this routine, without overloading it, by manipulating how the routine is called. The approach used in the hyper-dual version of Joe is to call the PETSc GMRES routine four times, solving for the real part, first-derivative components, and the second-derivative component sequentially.

For this work, Joe is primarily used to compute the coefficient of lift, C_L , coefficient of drag, C_D , and moment coefficient, C_M, for several subsonic, transonic, and supersonic airfoil shapes. The aerodynamic forces are computed using only the surface pressures; the viscous/shear stress contributions to the forces and moments are ignored. The hyper-dual number version of Joe is then used to compute the first and second derivatives of these aerodynamic quantities with respect to both free stream conditions and variables controlling the airfoil shape.

Results have been generated for a variety of airfoil shapes and free stream conditions. The results presented here are for a NACA 0012 airfoil, at a free stream Mach number of 0.8, at an angle of attack of 1°, and a Reynolds number of approximately 7 million. For these results the Spalart-Allmaras turbulence model was used. The Mach number variation around this airfoil is shown in Fig. 2(a). Figure 2(b) shows the residuals of the flow solution and the derivative calculations. As shown, the derivatives converge at the same rate as the flow solution. Table 1 shows a comparison between derivatives calculated using hyper-dual numbers and finite-difference approximations. In general, the values for the derivatives are in fairly good agreement. The first-derivative calculations compare very well, but the discrepancies are larger for the second-derivative calculations.

In the above example, the derivative values found using hyper-dual numbers are compared to values found using finite-differences (which include truncation and subtractive cancellation errors). Although the results are in fairly good agreement, the comparison does not prove the accuracy of the hyper-dual number calculations (which are supposed to be exact). In order to demonstrate the accuracy of the hyper-dual number calculations, the derivative values found using hyper-dual numbers need to be compared to known exact values.



(a) Mach number variation around the airfoil.

(b) Convergence history of the flow solver and derivative calculations.

Figure 2. Flow solution and convergence history of all calculations (including derivatives) for a NACA 0012 airfoil, Mach 0.8, $\alpha = 1°$, Re=7e6, using the Spalart-Allmaras turbulence model.

	C_L	$\frac{dC_L}{d\alpha}$	$\frac{dC_L}{dM}$
Joe, hyper-dual	0.165004436840803	9.299541767478981	0.644541632471284
Joe, finite-difference	0.165004436840802	9.299525259021113	0.644517820502788
	C_D	$\frac{dC_D}{d\alpha}$	$\frac{dC_D}{dM}$
Joe, hyper-dual	0.0554313191493638	0.3824569206943927	0.2828036900139284
Joe, finite-difference	0.0554313191493642	0.3824574885957777	0.2828030752632937
	C_M	$\frac{dC_M}{d\alpha}$	$\frac{dC_M}{dM}$
Joe, hyper-dual	5.66221208861855e-02	3.26456652828607e+00	5.39050871899535e-01
Joe, finite-difference	5.66221208861826e-02	3.26455810453763e+00	5.39038085348165e-01
	$\frac{d^2C_L}{d\alpha^2}$	$\frac{d^2C_L}{dM^2}$	$\frac{d^2C_L}{d\alpha dM}$
Joe, hyper-dual	-309.121862728704173	-445.302962067975045	-338.716668969934290
Joe, finite-difference	-310.051984087067524	-432.298641328543511	-331.287774990585206
	$\frac{d^2C_D}{d\alpha^2}$	$\frac{d^2C_D}{dM^2}$	$\frac{d^2C_D}{d\alpha dM}$
Joe, hyper-dual	13.7281574974742622	-7.8650754417178561	-5.7991795661637644
Joe, finite-difference	14.3322853585203820	-3.9510061888847763	-3.7844727351910028
	$\frac{d^2C_M}{d\alpha^2}$	$\frac{d^2C_M}{dM^2}$	$\frac{d^2C_M}{d\alpha dM}$
Joe, hyper-dual	-1.59088213769098e+02	-2.41278826062479e+02	-1.79210520986971e+02
Joe, finite-difference	-1.58375396130950e+02	-2.32758951002054e+02	-1.74041336897801e+02

Table 1. A comparison of first and second derivatives computed using hyper-dual numbers and finite-difference approximations. The values given are for a NACA 0012 airfoil at Mach 0.8, $\alpha = 1°$, Re= 7 million, using the Spalart-Allmaras turbulence model.

The example chosen for this demonstration is inviscid, supersonic flow over a wedge. Specifically, we will look at the derivatives of the pressure ratio across the oblique shock, $\frac{P_2}{P_1}$, with respect to the incoming Mach number. Although no explicit equation exists relating the Mach number and pressure after an oblique shock to the incoming Mach number (the oblique shock relation, Eq. (35) does provide this relationship *implicitly*), analytic derivatives can be derived using an adjoint approach.

The equation for the pressure ratio across an oblique shock is

$$\frac{P_2}{P_1} = 1 + \frac{2\gamma}{\gamma+1}\left(M_1^2 \sin^2\beta - 1\right). \tag{34}$$

In order to find the shock angle, β, for a given wedge angle, θ, the oblique shock relation,

$$\tan\theta = \frac{2\cot\beta\left(M_1^2\sin^2\beta - 1\right)}{M_1^2\left(\gamma + \cos\left(2\beta\right)\right) + 2}, \tag{35}$$

is solved iteratively. The equation for the first derivative of the pressure ratio across an oblique shock is

$$\frac{d\left(\frac{P_2}{P_1}\right)}{dM_1} = \frac{\partial\left(\frac{P_2}{P_1}\right)}{\partial M_1} + \psi_1 \frac{\partial R}{\partial M_1}, \tag{36}$$

and the equation for the second derivative is

$$\frac{d^2\left(\frac{P_2}{P_1}\right)}{dM_1^2} = \frac{\partial^2\left(\frac{P_2}{P_1}\right)}{\partial M_1^2} + \psi_1 \frac{\partial^2 R}{\partial M_1^2} + \psi_2 \frac{\partial R}{\partial M_1} + \psi_3 \left(\frac{\partial R}{\partial M_1}\right)^2. \tag{37}$$

Where there are three adjoint equations

$$\psi_1 = -\frac{\partial\left(\frac{P_2}{P_1}\right)}{\partial\beta}\left(\frac{\partial R}{\partial\beta}\right)^{-1}, \tag{38}$$

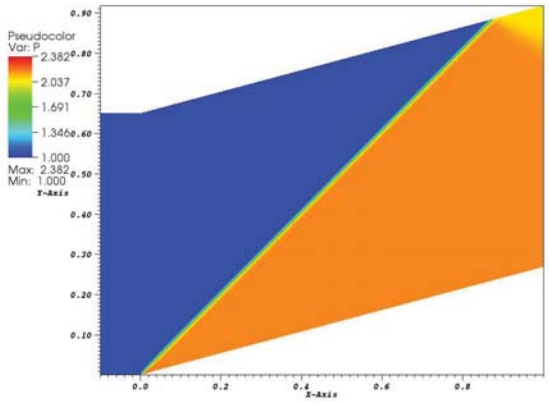
$$\psi_3 = -\left(\frac{\partial^2\left(\frac{P_2}{P_1}\right)}{\partial\beta^2} + \psi_1\frac{\partial^2 R}{\partial\beta^2}\right)\left(\frac{\partial R}{\partial\beta}\right)^{-2}, \tag{39}$$

$$\psi_2 = -2\left(\frac{\partial^2\left(\frac{P_2}{P_1}\right)}{\partial M_1\partial\beta} + \psi_1\frac{\partial^2 R}{\partial M_1\partial\beta} + \psi_3\frac{\partial R}{\partial M_1}\frac{\partial R}{\partial\beta}\right)\left(\frac{\partial R}{\partial\beta}\right)^{-1}, \tag{40}$$

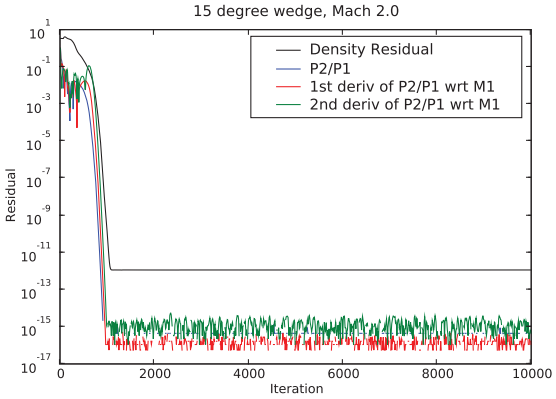
and the oblique shock relation is cast as the residual equation,

$$R = \frac{2\cot\beta\left(M_1^2\sin^2\beta - 1\right)}{M_1^2\left(\gamma + \cos\left(2\beta\right)\right) + 2} - \tan\theta = 0. \tag{41}$$

Figure 3(a) shows the CFD calculations for a Mach 2.0 flow over a 15° wedge. To allow for easier comparisons with finite-difference results, the pressure ratio is averaged over the center 60% of the wedge, from $x = 0.2$ to $x = 0.8$. These results are summarized in Table 2. The convergence history of the hyper-dual number calculations for the derivatives of the average pressure ratio is shown in Fig. 3(b). A detailed comparison between the hyper-dual CFD results and the exact values is given in Fig. 4. This figure shows the values of the pressure ratio, and its first and second derivative, at every point along the wedge. This figure also shows the exact solution and provides plots of the error for every point along the wedge. Although not exact to machine precision, the hyper-dual results are in good agreement with the exact solution over most of the wedge. From these plots, it can be seen that the error in the hyper-dual number derivative calculations along the wedge follows the same trends as the error in the pressure ratio calculation. This implies that the error in the derivative calculations may be related to the error of the flow solution, and might be reduced by performing the calculations on a finer mesh.

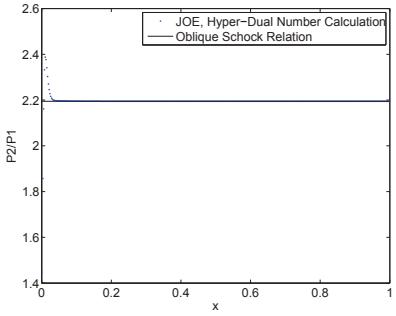


(a) Pressure Ratio.

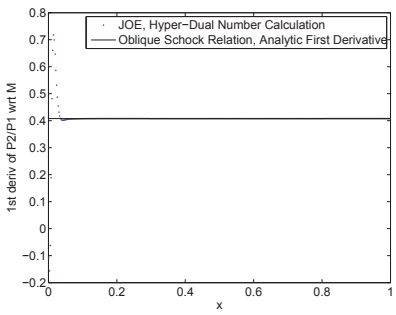


(b) Convergence history of the flow solver and derivative calculations.

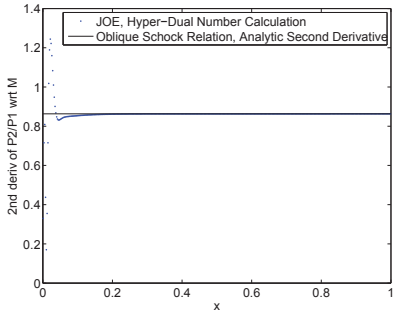
Figure 3. Flow solution and convergence history for inviscid Mach 2.0 flow over a 15° wedge.



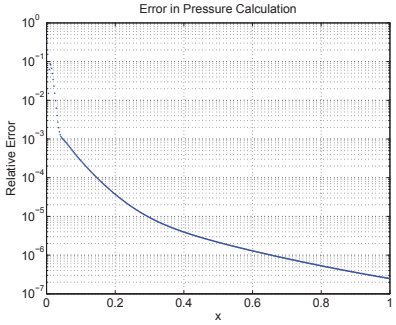
(a) Pressure Ratio, $\frac{P_2}{P_1}$.



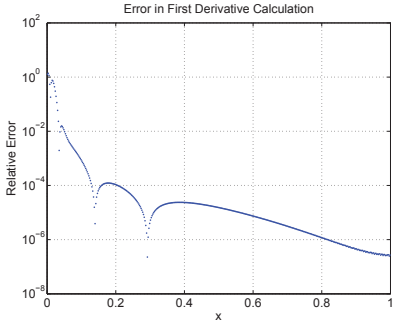
(b) First Derivative of $\frac{P_2}{P_1}$ wrt M_1.



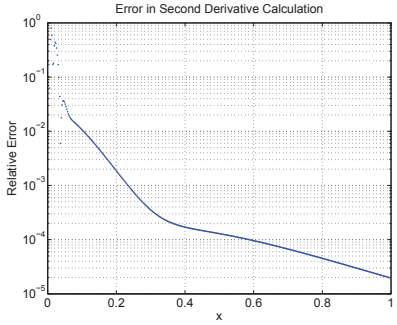
(c) Second Derivative of $\frac{P_2}{P_1}$ wrt M_1.



(d) Error in Pressure Ratio Calculation.



(e) First Derivative Calculation Error.



(f) Second Derivative Calculation Error.

Figure 4. Results from a hyper-dual CFD calculation are compared to the exact value and analytic derivatives obtained from the oblique shock relation, for a 15° wedge at Mach 2. In the top three plots, the exact values are represented by a black line. The blue dots represent the values from the CFD calculation for every control volume along the wedge. The bottom three plots show the error between the hyper-dual CFD calculation and the exact results.

	$\frac{P_2}{P_1}$	$\frac{d\left(\frac{P_2}{P_1}\right)}{dM_1}$	$\frac{d^2\left(\frac{P_2}{P_1}\right)}{dM_1^2}$
Oblique shock relation, adjoint	2.194653133607664	0.407667273032935	0.863026223964081
Oblique shock relation, hyper-dual	2.194653133607664	0.407667273033135	0.863026223952016
Joe, hyper-dual	2.194664703661337	0.407666379350755	0.862810467824695
Joe, finite-difference	2.194664703661338	0.407665948554126	0.864641691578072

Table 2. A comparison of first and second derivatives computed using analytic formulas, hyper-dual numbers, and finite-difference approximations. The values given are for an inviscid Mach 2.0 flow over a 15° wedge.

C. Computational Efficiency

Forming the gradient of $f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$, requires n first-derivative calculations. The forward-difference approximation requires $n+1$ function evaluations. The central-difference approximation requires $2n$ function evaluations. The complex-step approximation requires n complex function evaluations. Each complex function evaluation involves more work. Adding two complex numbers is equivalent to 2 real additions. Multiplying two complex numbers is equivalent to 4 real multiplications and 3 real additions. So a complex function evaluation should take about 2 to 4 times the runtime of a real function evaluation. The use of hyper-dual numbers requires n hyper-dual function evaluations. Adding hyper-dual numbers is equivalent to 4 real additions. Multiplying two hyper-dual numbers is equivalent to 9 real multiplications and 5 real additions. So a hyper-dual function evaluation should take about 4 to 14 times the runtime of a real function evaluation.

Forming the Hessian of $f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$, requires $\frac{n(n+1)}{2}$ second-derivative calculations. The forward difference approximation requires $(n+1)^2$ function evaluations. The central difference approximation requires $2n(n+1)$ function evaluations. The use of hyper-dual numbers requires $\frac{n(n+1)}{2}$ hyper-dual function evaluations, which as discussed above is equivalent to between $2n(n+1)$ and $7n(n+1)$ real function evaluations. So the use of hyper-dual numbers is 1 to 3.5 times as expensive as the central-difference approximation, which is itself about twice as expensive as the forward-difference approximation.

In many situations, such as numerical optimization using Newton's method, both the gradient and Hessian need to be computed. This does not change the results for the hyper-dual number method and forward-difference approximations. This is because the required function evaluations are used for both calculations. However, the central-difference approximation now requires $2n(n+2)$ function evaluations.

In practice, a hyper-dual CFD run takes roughly 10 times that of a real-valued CFD run. This is reasonable based on the discussion above. The actual value depends greatly on the compiler and optimization flags that are used. It has been observed to be as low as 7 or as high as 30, but the average is around 10. Computing the Hessian using hyper-dual numbers requires 5 times the time required to use forward-differences, and 2.5 times the time required to use central-differences. In some situations this can be greatly improved, especially when the analysis uses an iterative procedure.[15,16] When an iterative procedure is involved, rather than converging the procedure using hyper-dual numbers, the procedure can be converged using real numbers and then the derivatives can be found by running a single iteration using hyper-dual numbers. For instance, hyper-dual numbers were used to compute the gradient and Hessian for the multi-objective optimization of a supersonic aircraft using Newton's method. The two objectives were range[17] and the sonic-boom over-pressure at the ground.[18,19] This optimization initially took 7 times as long as using forward-differences and 3.6 times as long as using central-differences. After identifying a small iterative procedure, and making the changes discussed above, the runtime of the hyper-dual version of the code decreased by a factor of 8. This means that the runtime for the optimization using hyper-dual numbers was reduced to 0.9 times the forward-difference runtime and 0.46 times the central-difference runtime.

VI. Conclusion

This paper has demonstrated the difficulty of calculating accurate second-derivative information using finite-difference or complex-step approximations. This difficulty arose because, unlike for first derivatives, second-derivative complex-step approximations are subject to subtractive cancellation error. A technique for producing highly-accurate second derivatives is desired. One approach might be to use a standard automatic

or algorithmic differentiation tool to compute the second derivatives. Instead, the approach taken here was to attempt to find a number system capable of producing second-derivative calculations that are not subject to subtractive cancellation error. This led to the development of hyper-dual numbers, a higher-dimensional extension of dual numbers.

The traditional use of dual numbers has been as dual-quaternions,[20,21] with applications in robotics[22] and computer graphics. There has also been some use of dual numbers for calculating first derivatives.[23] A recursive formulation, using dual numbers with dual number components, to compute second and higher derivatives has also been developed,[24] and should be equivalent, in form and function, to the hyper-dual numbers developed here. Derivative calculations using hyper-dual numbers are free from both truncation error and subtractive cancellation error, making them exact. This also makes the use of hyper-dual numbers equivalent, in function, to automatic differentiation.

Hyper-dual numbers have been implemented as a class, using operator overloading, in C++ and MATLAB. The accuracy of these calculations has been demonstrated using a simple, analytic function and a large-scale CFD code. In situations where finite differencing is used to compute second derivatives, hyper-dual numbers can be used if greater accuracy is desired. The computational cost of using hyper-dual numbers to compute gradients and Hessians of functions of many variables may be prohibitive in some applications. In these situations, where adjoint methods are desired, the accuracy and ease of implementation of hyper-dual numbers may make them useful in developing and verifying the adjoint code. The computational cost of using hyper-dual numbers may be greatly reduced when the analyses involve iterative procedures.

Acknowledgments

This work was funded, in part, by the United States Department of Energy's Predictive Science Academic Alliance Program (PSAAP) at Stanford University.

References

[1]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The complex-step derivative approximation," *ACM Trans. Math. Softw.*, Vol. 29, No. 3, 2003, pp. 245–262.

[2]Martins, J. R. R. A., Kroo, I. M., and Alonso, J. J., "An Automated Method For Sensitivity Analysis Using Complex Variables," *AIAA paper 2000-0689, 38th Aerospace Sciences Meeting*, 2000.

[3]Martins, J. R. R. A., *A Coupled-Adjoint Method for High-Fidelity Aero-Structural Optimization*, Ph.D. thesis, Stanford University, Stanford, CA, October 2002.

[4]Hascoët, L. and Pascual, V., "TAPENADE 2.1 user's guide," Technical Report RT-0300, INRIA, 2004.

[5]Griewank, A., Juedes, D., and Utke, J., "Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++," *ACM Trans. Math. Softw.*, Vol. 22, June 1996, pp. 131–167.

[6]Kantor, I. and Solodovnikov, A., *Hypercomplex Numbers: An Elementary Introduction to Algebras*, Springer-Verlag, New York, 1989.

[7]Deakin, M. A. B., "Functions of a Dual or Duo Variable," *Mathematics Magazine*, Vol. 39, No. 4, 1966, pp. 215–219.

[8]Eastham, M. S. P., "2968. On the Definition of Dual Numbers," *The Mathematical Gazette*, Vol. 45, No. 353, 1961, pp. 232–233.

[9]Pecnik, R., Terrapon, V. E., Ham, F., and Iaccarino, G., "Full system scramjet simulation," *Annual Research Briefs, Center for Turbulence Research, Stanford University*, 2009.

[10]Lai, K. L. and Crassidis, J. L., "Extensions of the First and Second Complex-Step Derivative Approximations," *J. Comput. Appl. Math.*, Vol. 219, No. 1, 2008, pp. 276–293.

[11]Olariu, S., *Complex Numbers in N Dimensions*, Vol. 190 of *North-Holland Mathematics Studies*, North-Holland, Amsterdam, 2002.

[12]`http://adl.stanford.edu/`.

[13]Cervio, L. I. and Bewley, T. R., "On the extension of the complex-step derivative technique to pseudospectral algorithms," *Journal of Computational Physics*, Vol. 187, No. 2, 2003, pp. 544 – 549.

[14]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Connection Between The Complex-Step Derivative Approximation And Algorithmic Differentiation," *AIAA paper 2001-0921, 39th Aerospace Sciences Meeting*, 2001.

[15]Griewank, A. and Walther, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2008.

[16]Bartholomew-Biggs, M., "Using Forward Accumulation for Automatic Differentiation of Implicitly-Defined Functions," *Computational Optimization and Applications*, Vol. 9, 1998, pp. 65–84.

[17]Sobieszczanski-Sobieski, J., Altus, T. D., Phillips, M., and Sandusky, R., "Bi-level Integrated System Synthesis (BLISS) for Concurrent and Distributed Processing," *AIAA paper 2002-5409, 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Oprimization*, 2002.

[18]Thomas, C. L., "Extrapolation of Sonic Boom Pressure Signatures by the Waveform Parameter Method," NASA-TN-D-6832, June 1972.

[19]Carlson, H. W., "Simplified Sonic-Boom Prediction," NASA-TP-1122, March 1978.

[20]Clifford, W. K., "Preliminary Sketch of Biquaternions," *Proc. London Math. Soc.*, Vol. s1-4, No. 1, 1871, pp. 381–395.

[21]Hudson, R. W. H. T., "Review: Geometrie der Dynamen. Von E. Study," *The Mathematical Gazette*, Vol. 3, No. 44, 1904, pp. 15–16.

[22]Daniilidis, K., "Hand-Eye Calibration Using Dual Quaternions," *The International Journal of Robotics Research*, Vol. 18, No. 3, 1999, pp. 286–298.

[23]Leuck, H. and Nagel, H.-H., "Automatic Differentiation Facilitates OF-Integration into Steering-Angle-Based Road Vehicle Tracking," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, Vol. 2, 1999, pp. 2360.

[24]Piponi, D., "Automatic Differentiation, C++ Templates, and Photogrammetry," *journal of graphics, gpu, and game tools*, Vol. 9, No. 4, 2004, pp. 41–55.