cite peer-reviewed, updated version:
Lantoine, G., Russell, R. P., Dargent, T., “Using Multicomplex Variables for Automatic Computation of High-order Derivatives,”
ACM Transactions on Mathematical Software, Vol. 38, No. 3, Article 16, April 2012, 21 pages,
DOI = 10.1145/2168773.2168774, http://doi.acm.org/10.1145/2168773.2168774

AAS 10-218

USING MULTICOMPLEX VARIABLES FOR AUTOMATIC COMPUTATION OF HIGH-ORDER DERIVATIVES

Gregory Lantoine[*], **Ryan P. Russell**[†], **and Thierry Dargent**[‡]

The computations of the high-order partial derivatives in a given problem are in general tedious or not accurate. To combat such shortcomings, a new method for calculating exact high-order sensitivities using multi-complex numbers is presented. Inspired by the recent complex step method that is only valid for first order sensitivities, the new multi-complex approach is valid to arbitrary order. The mathematical theory behind this approach is revealed, and an efficient procedure for the automatic implementation of the method is described. Several applications are presented to validate and demonstrate the accuracy and efficiency of the algorithm. The results are compared to conventional approaches such as finite differencing, the complex step method, and two separate automatic differentiation tools. Our multi-complex method is shown to have many advantages, and it is therefore expected to be useful for any algorithm exploiting high-order derivatives, such as many non-linear programming solvers.

INTRODUCTION

SENSITIVITY analysis, i.e. computing the partial derivatives of a function with respect to its input variables, is often required in a variety of engineering problems. For instance, most optimization algorithms require accurate gradient and hessian information to find a solution efficiently.[1] In practice, accuracy, computational cost, and ease of implementation are the most important criteria when sensitivities must be evaluated.

There are many methods for generating the desired sensitivities. First, the partial derivatives can be analytically derived by hand, which is typically most accurate and efficient. However, for complicated problems, this can be a tedious, error-prone and time-consuming process. Numerical methods are therefore preferred in general. One classical numerical method is finite differencing that finds approximation formulas of derivatives by truncating a Taylor series of the function about a given point.[2] This technique is very simple to implement, but suffers from large roundoff errors, especially for high-order derivatives.[3]

Another numerical method is Automatic Differentiation (AD). Invented in the 1960s, AD is a chain rule-based evaluation technique for obtaining automatically the partial derivatives of a function.[4] AD exploits the fact that any function, no matter how complicated, can be expressed in terms of composition and arithmetic operations of functions with known derivatives. By applying the chain rule repeatedly to these elementary operations and functions, derivatives can be computed therefore automatically. Some AD tools are implemented by preprocessing the program that computes the function value. The original source code is then extended to add the new instructions that compute these derivatives. ADIFOR[5] and TAPENADE[6] represents successful implementations of this approach. Other AD tools, such as AD02,[7] ADOL-F[8] and OCEA,[9] keep the original program but use derived datatypes and operator overloading to compute the function value and its

[*]PhD Candidate, School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Dr., Atlanta, Georgia, 30318, USA, gregory.lantoine@gatech.edu.
[†]Assistant Professor, School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Dr., Atlanta, Georgia, 30318, USA, ryan.russell@gatech.edu.
[‡]Head of Platform and Satellite Research Group, Thales Alenia Space, 100 Bd du Midi, BP 99, Cannes la Bocca, 06322, France, thierry.dargent@thalesaleniaspace.com

differential at runtime. The major advantage of all these tools is that exact derivatives can be found automatically, however they generally have the drawback of being hard to implement and computationally intensive in terms of machine time and memory. They are also often limited to second- and in some cases first-order derivatives only.

Complex arithmetic can be another way to obtain accurate sensitivities. The use of complex numbers for the numerical approximation of derivatives was introduced by Lyness and Moler.[10,11] Relying on Cauchy's integral theorem, they developed a reliable method for calculating the n^{th} derivative of an analytic function from a trapezoidal approximation of its contour integral. Later Fornberg developed an alternative algorithm based on the Fast Fourier Transform.[3] However, both approaches are of little practical use because they require an excessive number of function evaluations to obtain a high accuracy. More recently, Squire and Trapp developed an elegant, simple expression based on a complex-step differentiation to compute first-order derivatives of an analytic function.[12] They pointed out that their method is accurate to machine precision with a relatively easy implementation. Therefore, this method is very attractive and since the 2000s it has been applied in an increasing number of studies in many fields.[13–17] A thorough investigation on the practical implementation of this method in different programming languages was also performed,[13,18] which makes now this technique very well understood. Note that contrary to what many authors imply, this method is not related to the other complex approach of Lyness and Moler, since the complex-step differentiation does not rely on the Cauchy integral theorem. In particular, one major difference is that the complex-step differentiation gives an expression for the first-order derivatives only, which limits greatly its range of applications. Several extension formulas to second-order derivatives have been published in the literature,[19,20] but they all suffer from roundoff errors.

In this paper, we describe a new way of computing second- and higher-order derivatives by using multicomplex numbers, a multi-dimensional generalization of complex numbers. By introducing a small perturbation into the appropriate multicomplex direction, higher-order sensitivities exact to machine precision can be retrieved directly from the results. As in the complex method, when the program can handle multicomplex algebraic operations, no special coding is required in the function calls as the higher-dimensional space carries the underlying problem sensitivities. Our multicomplex step differentiation (MCX) method therefore combines the accuracy of the analytical method with the simplicity of finite differencing.

Since standard multicomplex algebra is not built into existing mathematical libraries of common programming languages, an object-oriented multicomplex toolbox (coded both in Matlab and Fortran 90) is presented to encapsulate the new data types and extend operators and basic mathematic functions to multicomplex variables. By exploiting some properties of multicomplex numbers, an elegant recursive operator-overloading technique is derived to implement the overloading without much effort.

To our knowledge this is the first time multicomplex arithmetic is exploited to generate partial derivatives of any order. We can only mention the method of Turner who used quaternions (another extension of complex numbers) to compute all first derivative elements of functions of three variables with a single call.[21] However this method does not evaluate high-order derivatives.

This paper is organized as follows. First, we present the mathematical theory behind the multicomplex step differentiation. A review of the basic definition of multicomplex numbers is given, as well as the extension of the concepts of differentiability and holomorphism to multicomplex higher-dimensional space. Next, we investigate how to implement in practice our method in the common programming languages Fortran and Matlab. Finally, several applications and comparisons are presented to validate and demonstrate the performance of the multicomplex approach.

THEORY

In this section, the mathematical formalism associated to the multicomplex algebra is introduced. Definitions and basic properties of multi-complex numbers are briefly recalled. The natural multicomplex extension of differentiability and holomorphism is given. Then the multicomplex step differentiation is proved and explained in details with a simple numerical example.

Definition of Multicomplex numbers

There exist several ways to generalize complex numbers to higher dimensions. The most well-known extension is given by the quaternions invented by Hamilton,[22] which are mainly used to represent rotations in three-dimensional space. However, quaternions are not commutative in multiplication, and we will see later that this property prevents them from being a suitable for computing partial derivatives.

Another extension was found at the end of the 19^{th} century by Corrado Segre who described special multi-dimensional algebras and he named their elements 'n-complex numbers'.[23] This type of number is now commonly named a *multicomplex number*. They were studied in details by Price[24] and Fleury.[25]

To understand a multicomplex number, we can recall first the definition of the set of complex numbers, $\mathbb{C}$, which should be more familiar to the reader. $\mathbb{C}$ can be viewed as an algebra generated by the field of real numbers, $\mathbb{R}$, and by a new, non-real element I whose main property is $i^2 = -1$.

$$\mathbb{C} := \{x + yi \;/\; x, y \in \mathbb{R}\} \tag{1}$$

The same recursive definition applies to the set of multicomplex numbers of order n and defined as:

$$\mathbb{C}^n := \left\{z_1 + z_2 i_n \;/\; z_1, z_2 \in \mathbb{C}^{n-1}\right\} \tag{2}$$

where $i_n^2 = -1$, $\mathbb{C}^1 := \mathbb{C}$, $\mathbb{C}^0 := \mathbb{R}$.

This formula emphasizes the formal similarity of complex and multicomplex numbers. We will take advantage of this observation in the next section.

Other useful representations of multi-complex numbers can be found by repetitively applying Eq. 2 to the multi-complex coefficients of lower dimension. Decomposing z_1 and z_2 from Eq. 2, we obtain:

$$\mathbb{C}^n := \left\{z_{11} + z_{12} i_{n-1} + z_{21} i_n + z_{22} i_n i_{n-1} \;/\; z_{11}, z_{12}, z_{21}, z_{22} \in \mathbb{C}^{n-2}\right\} \tag{3}$$

In the end, it is clear (see Eq. 4) that we can represent each element of $\mathbb{C}^n$ with 2^n coefficients in $\mathbb{R}$: one coefficient x_0 for the real part, n coefficients $x_1, ..., x_n$ for the 'pure' imaginary directions, and additional coefficients corresponding to 'cross coupled' imaginary directions. We note that the cross directions do not exist in $\mathbb{R}$ or $\mathbb{C}$, but appear only in $\mathbb{C}^n$ for $n \geq 2$. For instance, to make the notation of Eq. 4 more clear, one can make the analogy between $i_1 i_2$ and the standard product of the imaginary directions i_1 and i_2, which implies that $(i_1 i_2)^2 = (i_1)^2 (i_2)^2 = (-1)(-1) = 1$ and $i_1 i_2 = i_2 i_1$.

$$\begin{aligned}\mathbb{C}^n := \{&x_0 + x_1 i_1 + ... + x_n i_n + x_{12} i_1 i_2 + ... + x_{n-1n} i_{n-1} i_n + ... + x_{1...n} i_1 ... i_n \\ &/\; x_0, ..., x_n, ..., x_{1...n} \in \mathbb{R}\}\end{aligned} \tag{4}$$

In addition, another way to represent multicomplex numbers is with matrices. In fact, it has been shown that every linear algebra can be represented by a matrix algebra.[26] One common example is the 2×2 matrix representation of complex numbers.[27] The following theorem extends this result to $\mathbb{C}^n$.

Theorem 1
Let matrix I_0 be the identity matrix. In addition, let matrices $I_1, ..., I_n$ be the matrix representations of the multicomplex imaginary basis elements $i_1, ..., i_n$ with the property $I_k^2 = -I_0$ for all $k \leq n$. These matrices can be constructed by recursion in the same way as the proof of this theorem in Appendix 1, after reordering the indices properly to be consistent with the representation of Eq. 4.

Then the set of $2^n \times 2^n$ real matrices of the form

$$M = x_0 I_0 + x_1 I_1 + ... + x_n I_n + x_{12} I_1 I_2 + ... + x_{n-1n} I_{n-1} I_n + ... + x_{1...n} I_1 ... I_n \tag{5}$$

is isomorphic to the multicomplex algebra $\mathbb{C}^n$. Thoses matrices are called Cauchy-Riemann matrices in the literature.[24] In other words, there's a one-to-one correspondence between Cauchy-Riemann matrices of this form and multicomplex numbers. Arithmetic operations $(+, -, x)$ on multicomplex numbers become then equivalent to arithmetic operations on their matrix representations. The proof of this theorem is given in Appendix 1.

In summary, we just reviewed three different representations of multicomplex numbers. We point out that the representations are not simply a matter of notational consequence. To the contrary, they will be essential to the development of the theory. To illustrate the various definitions, we consider several particular examples. First, we define the elements of $\mathbb{C}^2$, called bicomplex numbers. Among all the multicomplex numbers, they are the most known and studied, and have been used in several applications like fractals and quantum theory.[28,29] As shown in Eq. 6, a bicomplex number is composed of two complex numbers or four real numbers. It can also be represented by a 2×2 complex matrix or a 4×4 real matrix.

$$\begin{aligned}
\mathbb{C}^2 &:= \{z_1 + z_2 i_2 \;/\; z_1, z_2 \in \mathbb{C}\} \\
&:= \{x_0 + x_1 i_1 + x_2 i_2 + x_{12} i_1 i_2 \;/\; x_0, x_1, x_2, x_{12} \in \mathbb{R}\} \\
&\leftrightarrow \left\{ \begin{pmatrix} z_1 & -z_2 \\ z_2 & z_1 \end{pmatrix} \;/\; z_1, z_2 \in \mathbb{C} \right\} \\
&\leftrightarrow \left\{ \begin{pmatrix} x_0 & -x_1 & -x_2 & x_{12} \\ x_1 & x_0 & -x_{12} & -x_2 \\ x_2 & -x_{12} & x_0 & -x_1 \\ x_{12} & x_2 & x_1 & x_0 \end{pmatrix} \;/\; x_0, x_1, x_2, x_{12} \in \mathbb{R} \right\}
\end{aligned} \tag{6}$$

Another example is an element of $\mathbb{C}^3$, called a tricomplex number. As the dimensions of the corresponding matrices become unreasonably large, they are not given here. As shown in Eq. 7, a tricomplex number is composed of two bicomplex numbers, four complex numbers, or eight real numbers.

$$\begin{aligned}
\mathbb{C}^3 &:= \{z_1 + z_2 i_3 \;/\; z_1, z_2 \in \mathbb{C}^2\} \\
&:= \{z_{11} + z_{12} i_2 + z_{21} i_3 + z_{22} i_2 i_3 \;/\; z_{11}, z_{12}, z_{21}, z_{22} \in \mathbb{C}\} \\
&:= \{x_0 + x_1 i_1 + x_2 i_2 + x_3 i_3 + x_{12} i_1 i_2 + x_{13} i_1 i_3 + x_{23} i_2 i_3 + x_{123} i_1 i_2 i_3 \\
&\qquad / \; x_0, x_1, x_2, x_3, x_{12}, x_{13}, x_{23}, x_{123} \in \mathbb{R}\}
\end{aligned} \tag{7}$$

Finally, one last property of importance is that multicomplex addition and multiplication are associative and commutative, contrary to quaternions. In fact, from Eq. 4, the product of two elements of $\mathbb{C}^n$ is obtained by multiplying those two elements as if they were polynomials and then using the relations $i_k^2 = -1$. However, contrary to the complex numbers, the multicomplex numbers do not form a ring since they contain divisors of zero (the product of two non-zero multicomplex numbers can be equal to zero). This can be an issue for numerical computations as unexpected zeroed results may be generated when two divisors of zero happen to be multiplied together. In his book,[24] Price shows that those divisors have a very specific form (see Appendix 2), and are therefore extremely unlikely to be encountered in practice.

Holomorphic Functions

We recall now the notion of differentiability and holomorphicity in multicomplex analysis. This is a natural next step, since the power of multicomplex numbers in computing derivatives cannot be exploited until a full theory of multi-complex holomorphic functions is developed. For this discussion we will rely essentially on the work of Price.[24] We give the definitions of multicomplex differentiability and holomorphism, and we present a theorem that will be necessary for the derivation of the formulas of multicomplex step differentiation.

Definition 1
A function $f : \mathbb{C}^n \to \mathbb{C}^n$ is said to be multicomplex differentiable at $z_0 \in \mathbb{C}^n$ if the limit

$$\lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0} \tag{8}$$

exists. This limit will be called the first derivative of f at z_0 and will be denoted by $f'(z0)$.

Definition 2
A function f is said to be holomorphic in a open set $U \subset \mathbb{C}^n$ if $f'(z)$ exists for all $z \in U$.

This definition is not very restrictive, most usual functions are holomorphic in $\mathbb{C}^n$. Examples of non-holomorphic functions are the modulus and absolute value functions at zero.

Theorem 2
Let $f : U \subset \mathbb{C}^n \to \mathbb{C}^n$ be a function, and let also $f(z_1 + z_2 i_n) = f_1(z_1, z_2) + f_2(z_1, z_2) i_n$ where $z_1, z_2 \in \mathbb{C}^{n-1}$. The following three properties are equivalent:

1. f is holomorphic in U.
2. f_1 and f_2 are holomorphic in z_1 and z_2 and satisfy the multicomplex Cauchy-Riemann equations:
$$\frac{\partial f_1}{\partial z_1} = \frac{\partial f_2}{\partial z_2} \text{ and } \frac{\partial f_2}{\partial z_1} = -\frac{\partial f_1}{\partial z_2} \tag{9}$$
3. f can be represented, near every point $z_0 \in U$, by a Taylor series.

This theorem can be obtained from results in Reference 24. The equivalencies $(1) = (2)$ and $(1) = (3)$ were stated and proved in Theorem 24.2 and Theorem 27.1 respectively only for the special case $n = 2$ (bicomplex functions). Nevertheless, the same methods used can be employed to prove the theorem in the general case.

Multicomplex Step Differentiation

We now proceed to the main purpose of the paper. Relying on the third property of theorem 2, Taylor series expansions are performed and used to analytically demonstrate that the introduction of perturbations along multicomplex imaginary directions allows us to recover the partial derivatives of any order of a holomorphic function.

To facilitate the addition of perturbations along imaginary directions, we use the multicomplex representation of Eq. 4. For convenience, we must also define a new imaginary function that retrieves the real coefficient of a specified imaginary part of a multicomplex number.

Definition 3
Let $z \in \mathbb{C}^n$ be given by Eq. 4. The function $Im_{\sigma_k\{1,...,n\}} : \mathbb{C}^n \to \mathbb{R}$ is defined to be:

$$Im_{\sigma_k}(z) = x_{\sigma_k} \tag{10}$$

where $\sigma_k = \sigma_k\{1,...,n\}$ are all the combinations of the $\{1,...,n\}$ set of the following form:

$$\sigma_k\{1,...,n\} = \underbrace{1...1}_{k_1 \text{ times}} \; ... \; \underbrace{n...n}_{k_n \text{ times}} \tag{11}$$

for $k_1 \in \{0,1\}$, ..., $k_n \in \{0,1\}$, and $k_1 + ... + k_n = k \leq n$. For instance, for $n = 3$, $\sigma_3\{1,2,3\} = \{123\}$, $\sigma_2\{1,2,3\} = \{12, 13, 23\}$ and $\sigma_1\{1,2,3\} = \{1,2,3\}$.

To introduce our main result, for simplicity we start first with a function of one variable only. We demonstrate how to obtain the n^{th}-order derivative. Let $f : U \subset \mathbb{C}^n \to \mathbb{C}^n$ be a holomorphic function in U. Then from theorem 2, f can be expanded in a Taylor series about a real point x as follows:

$$\begin{aligned} f(x + hi_1 + ... + hi_n) &= f(x) + (i_1 + ... + i_n)hf'(x) + (i_1 + ... + i_n)^2 h^2 \frac{f''(x)}{2} + ... \\ &+ (i_1 + ... + i_n)^n h^n \frac{f^{(n)}(x)}{n!} + (i_1 + ... + i_n)^{n+1} h^{n+1} \frac{f^{(n+1)}(x)}{(n+1)!} + O(h^{n+2}) \end{aligned} \tag{12}$$

From the multinomial theorem,

$$(i_1 + ... + i_n)^k = \sum_{\substack{k_1,...,k_n \\ k_1+...+k_n=k}} \frac{n!}{k_1!...k_n!} i_1^{k_1} ... i_n^{k_n} \tag{13}$$

We focus on the single term on the right hand side of Eq. 13 containing the product $i_1...i_n$ of each of the imaginary directions (corresponding to the last imaginary component in Eq. 4). Since $i_k^2 = -1$ for $k = 1...n$, it is clear that the only possibility to obtain such a term is to have $k_1 = 1, ..., k_n = 1$ ($k_p = 0$ where $p = 1...n$ means that i_p is not present, and $k_p = 2$ will make i_p disappear as well since $i_p^2 = -1$). This combination is only allowed in the $(i_1 + ... + i_n)^n$ term. In fact, for $(i_1 + ... + i_n)^k$ where $k < n$, one of the k_p's in Eq. 13 must be equal to zero and for $(i_1 + ... + i_n)^{n+1}$, one of the k_p's must be greater than 1.

From Eq. 12, if we ignore terms $O(h^{n+2})$ we can see that the $(i_1 + ... + i_n)^n$ term is the only one associated to the n^{th}-order derivative $f^{(n)}$. Since the $i_1...i_n$ product uniquely appears in $(i_1 + ... + i_n)^n$, we can deduce that the real coefficient of the $i_1...i_n$ imaginary direction is a function of the n^{th}-order derivative $f^{(n)}$ only (i.e. no other derivatives involved). We can take advantage of this result by applying to both sides of Eq. 12 the imaginary function corresponding to the $i_1...i_n$ product (see Eq. 10). Noting that $\frac{n!}{1!...1!} = n!$ and dividing both sides by h^n, we get an expression of $f^{(n)}(x)$ with approximation error $O(h^2)$:

$$f^{(n)}(x) = \frac{Im_{1...n}(f(x + hi_1 + ... + hi_n))}{h^n} + O(h^2) \tag{14}$$

For a small step size h, this expression can be approximated by:

$$f^{(n)}(x) \approx \frac{Im_{1...n}(f(x + hi_1 + ... + hi_n))}{h^n} \tag{15}$$

It is easy to extend this result to obtain the n^{th}-order partial derivatives of any holomorphic function of p

variables:

$$\frac{\partial f^n(x_1,...,x_p)}{\partial x_1^{k_1}...x_p^{k_p}} \approx \frac{Im_{1...n}(f(x_1+h\sum_{j\in\Pi_1} i_j,...,x_p+h\sum_{j\in\Pi_p} i_j))}{h^n}$$

$$\text{where } \sum_{j=1}^{p} k_j = n, \Pi_j = \emptyset \text{ if } k_j = 0, \Pi_j = \left\{\sum_{l=1}^{j-1} k_l + 1, ..., \sum_{l=1}^{j} k_l\right\} \text{ if } k_j > 0 \tag{16}$$

which we will call the *multicomplex step derivative approximation*. Notice that this estimate is not subject to subtractive cancellation error, since it does not involve any difference operation, contrary to finite differencing. From the same single function call, it is also possible to retrieve the corresponding low-order partial derivatives.

$$\frac{\partial f^k(x_1,...,x_p)}{\partial x_1^{k'_1}...x_p^{k'_p}} \approx \frac{Im_{\sigma_{k'_1}\{\Pi_1\}...\sigma_{k'_p}\{\Pi_p\}}(f(x_1+h\sum_{j\in\Pi_1} i_j,...,x_p+h\sum_{j\in\Pi_p} i_j))}{h^k}$$

$$\text{where } \sum_{j=1}^{p} k'_j = k < n,\ k'_j \leq k_j \tag{17}$$

For example, the particular formulas to compute the full Hessian of a function of two variables are the following:

$$\frac{\partial f^2(x,y)}{\partial x^2} \approx \frac{Im_{12}(f(x+hi_1+hi_2,y))}{h^2} \tag{18a}$$

$$\frac{\partial f^2(x,y)}{\partial y^2} \approx \frac{Im_{12}(f(x,y+hi_1+hi_2))}{h^2} \tag{18b}$$

$$\frac{\partial f^2(x,y)}{\partial xy} \approx \frac{Im_{12}(f(x+hi_1,y+hi_2))}{h^2} \tag{18c}$$

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{Im_1(f(x+hi_1+hi_2,y))}{h} = \frac{Im_2(f(x+hi_1+hi_2,y))}{h} \tag{18d}$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{Im_1(f(x,y+hi_1+hi_2))}{h} = \frac{Im_2(f(x,y+hi_1+hi_2))}{h} \tag{18e}$$

Finally note that these results are not possible using quaternions or any non-commutative extension of complex numbers. In such cases, the multinomial theorem of Eq. 13 fails and the $i_1...i_n$ imaginary coefficient vanishes. For instance, since the imaginary units i, j, k of quaternions are anti-commutative under multiplication ($ij = -ji = k$), we have $(i+j)^2 = -2$, which is a real number only.

Simple Numerical Example

To illustrate Eq. 16 and Eq. 17, we consider the following standard holomorphic test function that many authors have previously used:[10,12,13,19,20]

$$f(x) = \frac{e^x}{\sqrt{\sin x^3 + \cos x^3}} \tag{19}$$

The exact first-, second-, and third-order derivatives at $x = 0.5$ are computed analytically and compared to the results given by the multicomplex step, the hybrid finite difference complex-step scheme developed by Lai,[19] and the central finite-difference formulas for step sizes in the range $10^{-100} \leq h \leq 1$. Since Lai does not give a formula for third-order derivatives, we derived our own approximated expression by applying Taylor series expansions on several complex perturbation steps:

$$f^{(3)}(x) \approx \frac{Im\,(f(x+ih) - f(x+2ih) - f(x-ih))}{h^3} \tag{20}$$

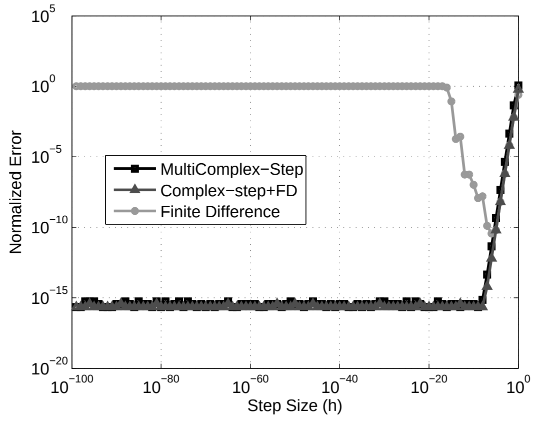

Figure 1. Normalized error: first-derivative

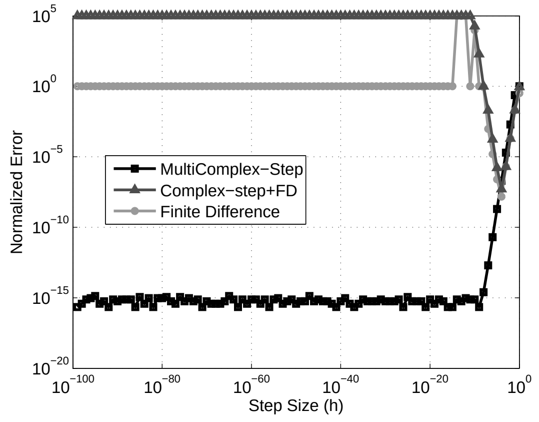

Figure 2. Normalized error: second-derivative

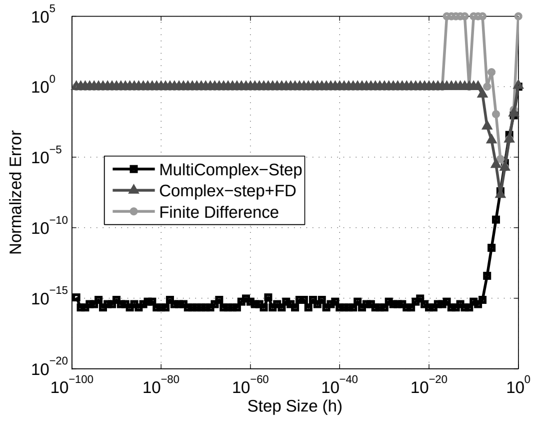

Figure 3. Normalized error: third-derivative

The multicomplex step method is exact to machine precision for both first and second-order derivatives with step sizes below 10^{-8}. In addition, since the MCX approach is not subjected to subtraction cancellations, we can choose extremely small step sizes with no loss of accuracy. As expected, for first-order derivatives our method gives identical results as the complex-step method while outperforming the central difference scheme. However, for higher-order derivatives, the complex-step method and central differences both suffer from subtraction errors. Note that in those cases the accuracy improvement of the complex-step method over finite differences is negligible. It was even observed that the formula given by Lai is not numerically stable as its associated error goes to infinity (not shown on the plots to preserve similar scales).

Finally, we observe that finite precision arithmetic imposes a practical lower limit on the step size h, and consequently an upper limit on the order of the derivative calculation. In fact, when double precision numbers are used, the smallest non-zero number that can be represented is 10^{-308}, and h^n must be therefore greater than this number to prevent underflow in Eq. 16: $h^n > 10^{-308}$. Also, h must be small since the error of the approximation in Eq. 16 is on the order of h^2 (see Eq. 14). To maintain approximate machine precision, $h^2 < \epsilon \approx 10^{-16}$. It follows then for double precision arithmetic:

$$n < \frac{log(10^{-308})}{log(10^{-8})} \approx 38 \tag{21}$$

In order to further prevent the underflow situation, it is also necessary to keep some margin to take into account the inherent dynamical magnitude excursion of internal variables during function evaluation. Therefore, it may be unreasonable to expect high precision with $n = 38$. Note that a complex number with $n = 35$ is represented with $2^{35} > 10^{10}$real numbers. Even modern computers with extraordinary memory capacity will not have enough storage to operate on multicomplex function calls in dimensions as high as 35.

The limitation of the step size h is illustrated in Figure 4 where computation errors of the test function (Eq. 19) up to the 6^{th}-order derivative are calculated for step h from 10^{-70} to 1.

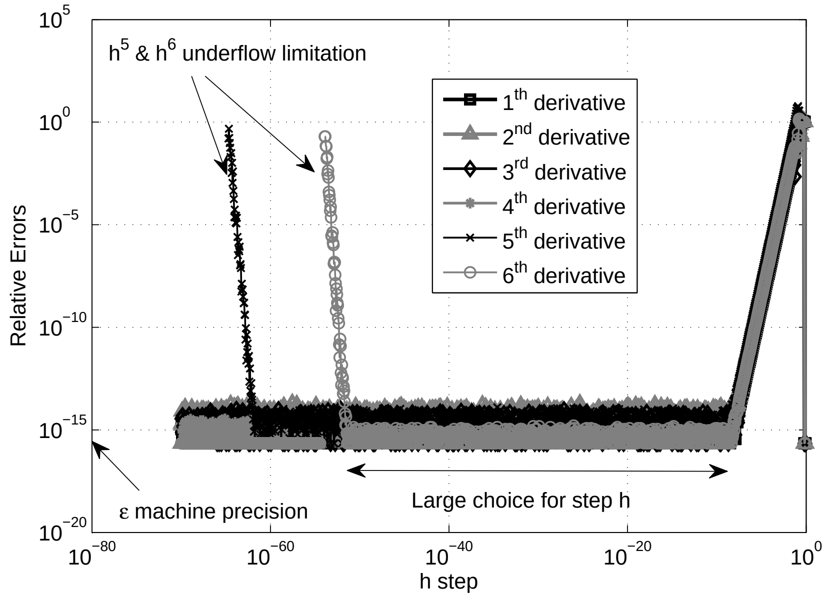

Figure 4. 1^{st} **to** 6^{th}**-order derivative relative errors**

IMPLEMENTATION

We now describe in details how to implement in practice the formulas given in the previous section. For completeness, computer details are discussed within two different types of programming frameworks: the compiled language Fortran for its speed, and the interpreted language Matlab for its ease of use. Of course, nothing is preventing the multicomplex adaptation to other languages like C++ or Java.

The objective is to develop a separate module or toolbox to support multicomplex arithmetic. The two main required capabilities are: 1) define derived datatypes to represent multicomplex variables, and 2) overload operators and intrinsic functions for allowing usual operations.

Implementation of multicomplex variables

The first step is to define the multicomplex variables. From Eq. 2, we choose the recursive data structure where a multicomplex variable of order n is composed of two multicomplex variables of order $n-1$.

$$z = \{z_1, z_2\} \tag{22}$$

In addition to being valid for any order n, another advantage of this structure is its convenience regarding extensions of operators and functions as we shall see in the next subsection.

In Matlab, this structure can be directly declared as recursive using a class statement, so only one definition is needed to include multicomplex numbers of any order. In the structure an additional integer field permits the retrieval of the order of the multicomplex variable. On the other hand, Fortran cannot handle recursive structures, so one definition per order is necessary. For instance, for multicomplex numbers of order 2 and 3, the syntax in Fortran is the following:

```
{Bicomplex number definition}
type bicomplex
  double complex :: z1
  double complex :: z2
end type
{Tricomplex number definition}
type tricomplex
  type(bicomplex) :: z1
  type(bicomplex) :: z2
end type
```

Additionally, in order to implement Eq. 16 and apply imaginary perturbation steps, it is also necessary to decompose a multicomplex variable into strictly real coefficients of its individual imaginary components. Furthermore, the left side of Eq. 16 requires the implementation of the Imaginary function Im() that extracts desired individual imaginary elements. To satisfy those two requirements, we require a mapping from the current recursive representation of Eq. 2 to that of the real coefficient representation of Eq. 4. This conversion can be deduced from the simple tree below that decomposes successively one multicomplex number into into two multicomplex numbers of lower order.

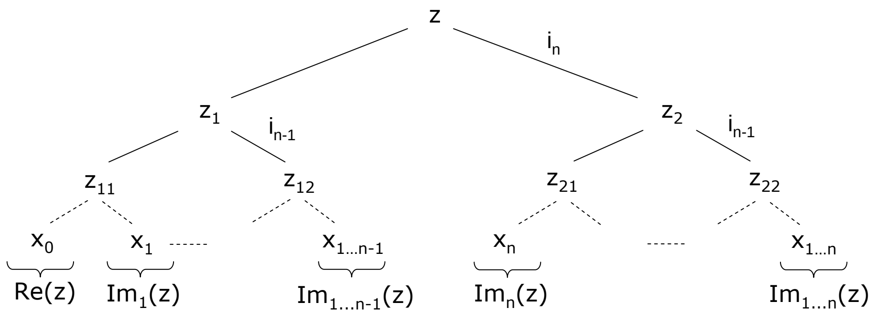

Figure 5. Tree Representation of a multicomplex number of order n.

Moving all the way down the tree allows us to locate one specific imaginary element from a multicomplex number, which serves as a basis of implementation for the Im function. In the same way, by traversing up the tree, we define a multicomplex variable according to all its imaginary components and associated real coefficients.

Operator and Function Overloading

It is necessary to redefine operational rules so that they can take multicomplex numbers as arguments. This procedure is called *overloading*. This should involve basic math operations ($+$,$-$,$\times$,$/$,$\hat{}$), relational logic operators ($<$,$>$,$==$), standard library functions (sin, $asin$, exp, ln, ...), and linear algebra operations (matrix-vector operations, matrix inversion, eigenvalue computations).

Basic functions and operations The recursive multicomplex representation we selected in the previous subsection makes the extension of any operation and function definition quite simple. In fact, with this representation, it turns out that arithmetic operations on multicomplex numbers are completely identical to respective operations on complex ones. For example, the multiplication of two multicomplex numbers has the following form:

$$z \times w = (z_1 + z_2 i_n)(w_1 + w_2 i_n) = (z_1 w_1 - z_2 w_2) + (z_1 w_2 + z_2 w_1) i_n \tag{23}$$

which is the exact same form as the complex multiplication. The same property can be observed for any other function or operation. This result can be readily deduced from Eq. 2 where it is clear that multicomplex numbers have the same general form as complex numbers. Since $i_n^2 = -1$ in Eq. 2, operation rules will be the same for complex and multicomplex numbers. It follows that we can re-use the same existing complex number overloading routines for complex numbers. The only change required is to replace the complex datatype with the corresponding multicomplex datatype. This results in a very elegant and simple implementation.

Relational logic operators Regarding relational logic operators, we decided to follow the same strategy as Martins.[13] These operators are usually used inside conditional statements which may lead to different execution branches. To compute correct derivatives, the execution branch should stay the same whether the calculations are in made with real or multicomplex numbers. It follows that only the real parts of the arguments should be compared.

Linear algebra Enabling linear algebra capabilities demands extra care. Here, at least two strategies are possible. First, linear algebra algorithms can be re-written to support multicomplex arguments. However, this can be a tedious process. For instance, in Fortran, linear algebra routines are commonly provided by the LAPACK package which consists of hundreds of cryptic separate routines (for Gaussian elimination, LU factorization, QR factorization ...). In Matlab the situation is even worse as linear algebra routines are built-in and cannot be accessed.

A second strategy is to take advantage of the matrix representation of Eq. 5. By mapping multicomplex variables to higher-dimensional real- or complex-valued matrices, we can use directly existing real or complex built-in algorithms, at the expense of memory usage. For instance, to solve the multicomplex linear matrix equation $Az = b$ where $A \in \mathbb{C}^{n\ p\times p}$, $z \in \mathbb{C}^{n\ p\times 1}$, $b \in \mathbb{C}^{n\ p\times 1}$, we can carry out the following transformation of A:

$$A \leftrightarrow M = A_0 I_0 + A_1 I_1 + ... + A_n I_n + A_{12} I_1 I_2 + ... + A_{n-1n} I_{n-1} I_n + ... + A_{1...n} I_1 ... I_n \tag{24}$$

where $A_0, ..., A_{1...n} \in \mathbb{R}^{p\times p}$ and the expressions for the $I_k's$ are given in theorem 1 (note that in the formula the $1's$ represent real identity matrices of dimension $p \times p$). The matrix equation becomes:

$$\underbrace{M}_{2^n p\times 2^n p} \underbrace{\begin{pmatrix} x_0 \\ \vdots \\ x_{1...n} \end{pmatrix}}_{2^n p\times 1} = \underbrace{\begin{pmatrix} b_0 \\ \vdots \\ b_{1...n} \end{pmatrix}}_{2^n p\times 1} \tag{25}$$

where $x_0, ..., x_{1...n} \in \mathbb{R}^{p\times 1}$ and $b_0, ..., b_{1...n} \in \mathbb{R}^{p\times 1}$.

Eq. 25 is solved as a real-valued matrix equation for $x_0, ..., x_{1...n}$. We can follow the exact same approach for other linear algebra algorithms like matrix inversion and eigenvalue problems. A similar strategy is used by Turner in his quaternion toolbox.[30]

Overall Procedure

The multicomplex step differentiation procedure can be summarized as follows:

1. Convert the function code to support multicomplex arithmetic. In Matlab, no change in the code is necessary and the user just needs to include the multicomplex toolbox in the path. However, in a Fortran code, all real types of the independent variables should be substituted with multicomplex declarations. In addition, if the value of any intermediate variable depends on the independent variables via assignment or via argument association, then the type of those variables must be changed to multicomplex as well (an easier yet memory inefficient option is to declare all variables multicomplex). The user enables overloading by simply inserting a ‘use module’ command for the module that contains all the multicomplex extensions and definitions. All these manipulations imply obviously that the user must have access to the source code that computes the value of the function. To avoid having several versions of the same code supporting different types of variables, one can use a preprocessor that automatically produces a single code that can be chosen to be real or multicomplex at compilation.

2. Apply small perturbation steps to the imaginary directions of the desired independent variables and compute the resulting function value.

3. Retrieve the corresponding partial derivatives using Eq. 16 and Eq. 17.

4. Repeat steps 3-4 for all variables and all partial derivatives.

For a real-valued function of p variables, this technique requires p^n function evaluations to compute all the partial derivatives up to n^{th} order, compared to $(np+1)^n$ and $(2np+1)^n$ function evaluations respectively for forward and central differences. If symmetries are considered, the number of function evaluations can be reduced by almost half (this is also true for finite differencing). More generally, if the sparsity pattern of the partial derivatives is known, our approach allows us to compute only the relevant non-zero components to save compute time.

In summary, our method shares with automatic differentiation the capability of computing systematically accurate partial derivatives with respect to desired input variables. However, a major difference is that the user has control on which components they want to compute by applying a perturbation step on specific imaginary directions and computing a series of multicomplex function evaluations. From that point of view, our approach is close to finite differences. Therefore, the multicomplex method could be classified as *semi-automatic differentiation*.

APPLICATIONS

Three examples of derivative-based applications illustrate the multicomplex step technique. The first one is a formal benchmark example. The next two are practical applications from the astrodynamics area. In all cases we compare the accuracy and computational cost between our approach and current other approaches, namely analytical differentiation, automatic differentiation and finite differences. In this section all the computations are performed on a PC in Fortran 90 with an optimization level of 2. Two different automatic differentiation tools are selected for the numerical comparisons:

- the new package AD02 from the HSL library. This tool relies on operator overloading to carry along derivative computations to the arithmetic operators and intrinsic functions.[7] It is one of the only automatic differentiation tools that allows the computation of high-order derivatives (i.e. order greater than two).

- the TAPENADE software, developed at INRIA (Institut National de Recherche en Informatique et Automatique). Contrary to AD02, it is a source transformation tool. Given a source program, this tool returns the first-order differentiated program.[6] By applying TAPENADE several times, the code for higher-order derivatives can be obtained as well. Note that this method for computing higher-order derivatives is somewhat tedious and not optimal from a complexity point of view.

We caution readers not to necessarily take the following results as an indication of the performance that could be expected on their code. Specificities of the problems play an important factor and some tools might perform better or worse depending on the applications.

Simple Mathematical Function

First, to check the correct implementation of the multicomplex method, the same simple one-dimensional function of section (Eq. 19) is used for the comparisons. Sensitivities up to third order are computed. From Figure 3, the multicomplex and finite difference derivatives are computed using a step size of 10^{-40} and 10^{-4} respectively. The analytical expressions of the derivatives are found with the help of the algebraic manipulation software, Maple, and optimized by introducing temporary intermediate variables to eliminate redundant computations. Table 1 summarizes the results of the comparison.

Table 1. Simple function example.

Method	3^{rd}-order derivative	Max. relative difference	Relative computational time
Analytical	-9.33191003819869	0.0	1.0
MultiComplex Step	-9.33191003819869	1.9 10^{-16}	37
AD02	-9.33191003819869	3.8 10^{-16}	149
TAPENADE	-9.33191003819869	3.8 10^{-16}	8
Finite Differences	-9.33197963348675	7.4 10^{-6}	3.5

Among methods that provide exact derivatives, the analytical and TAPENADE methods are by far the fastest. This is explained by the fact that they produce dedicated optimized code for the derivatives. However some implementation work to obtain the executable programs for computing the derivatives is necessary for those two methods. While this effort is significant for the analytical approach, this preliminary step is straightforward in the TAPENADE case as the source code needs only to be processed by the tool without any changes. On the other hand, the multicomplex and AD02 methods require very little change in the source code, but are slower for this simple test case. We point out that the multicomplex method is less computational intensive than AD02, which shows the advantage of our method among overloading techniques. Finally, we can say that finite difference is fast, but exhibits very poor accuracy. In this particular example, the small difference in computational speed between the analytical and finite difference cases is explained by the fact that the analytical expressions of the derivatives are quite complicated in comparison to the function (Eq. 19) alone.

Gravity Field Derivatives

For this example, the first-, second- and third-order partial derivatives of the gravitational potential of a non-spherical body with respect to cartesian coordinates are considered. These sensitivities are important for solving a variety of problems in satellite geodesy and navigation. For instance, the gravitational acceleration at any given location is obtained by computing the gradient of the potential. This acceleration is required for accurate numerical integration of satellite orbits. Additionally, the second- and third-order derivatives can be used in a variety of targeting or optimization problems that arise in spacecraft guidance and navigation.

The analytical method we employ is based on the classical spherical harmonic formulation where the derivatives are formed by exploiting recurrence relations on Legendre polynomials.[31] Finite differencing is not considered for this example as we saw in Figure 3 that its accuracy is extremely poor for high-order derivatives.

We use a 20 $\times$ 20 lunar gravity field model taken from GLGM-2 data,[32] which corresponds to 440 terms. The position vector in cartesian coordinates where the derivatives are estimated is $(x, y, z) = (2000, 0, 0)$ km, which corresponds to an altitude of about 300 km from the surface of the Moon. A step value of $h = 10^{-40}$ is taken in our multicomplex method. We use tricomplex numbers since derivatives are computed up to the third-order.

The resulting accuracy and computational comparison is made in Table 2. A sample of the third-order derivatives (corresponding to the (1,1,1) index) produced by each method is given, as well as relative computational time and maximum relative difference of all partial derivatives with respect to the analytical expressions. We know that the potential is a solution to Laplace's equation. Then, in cartesian coordinates, $\nabla^2 U = U_{xx} + U_{yy} + U_{zz} = 0$. A good indicator of the accuracy of each method is therefore the deviation from zero of the corresponding Laplacian.

Table 2. Lunar gravitational potential example.

Method	Sample 3^{rd}-order Sensitivity	Laplacian	Max difference	Relative comp. time
Analytical	$-4.23954197230525\underline{3}\ 10^{-12}$	$-8.3\ 10^{-25}$	0.0	1.0
MultiComplex Step	$-4.23954197230525\underline{0}\ 10^{-12}$	$-4.1\ 10^{-25}$	$6.0\ 10^{-15}$	20.9
AD02	$-4.23954197230525\underline{5}\ 10^{-12}$	$-1.1\ 10^{-24}$	$2.9\ 10^{-15}$	154.9
TAPENADE	$-4.23954197230525\underline{7}\ 10^{-12}$	$-1.6\ 10^{-24}$	$2.6\ 10^{-15}$	30.1

As expected, the analytical method is by far the fastest. Its implementation relies on a very efficient use of recurrence relations to reduce as much as possible the amount of computations. Therefore this method is very specific and not representative of the general situation (see next example for instance). It is included here only to provide a benchmark as any other method is likely to be far slower. Taking that into account, we can see that multicomplex step method is also accurate to machine precision while being reasonably fast (only one order of magnitude slower). In comparison, AD02 produced very accurate estimates, but it was more computational intensive, more than seven times slower than the multicomplex method. In this example it is apparent that the computational overhead of AD02 is again significantly larger than that of the multi-complex method. Finally, contrary to the previous simple example, TAPENADE is also slower than the multicomplex method. This may come from the multidimensional aspect of the problem as TAPENADE does not take advantage of the symmetries and computation redundancies of higher-order derivatives (TAPENADE is designed to produce first-order derivative code only.).

Trajectory State Transition Matrix

Another application is presented for a low-thrust spacecraft trajectory problem where the multicomplex approach is used to generate first- and second-order state transition matrices.[33] The trajectory model consists of an orbiting satellite subject to the Sun gravitational force and a constant inertial thrust. This kind of dynamical model often occurs in direct optimization methods when a low-thrust trajectory is divided into several segments of constant thrust.[34] To optimize the resulting set of thrust variables, partial derivatives of the final state vector with respect to the initial state vector of a given segment are usually required to help the solver converge toward an optimum. These sensitivities are the components of the so-called State Transition Matrices which map derivatives from one time to another on a given continuous trajectory.[35] Because such optimization problems are highly non-linear in nature, it is recommended to compute accurate first-and second-order derivatives to enable robust convergence.[36] The objective of this example is therefore to compute the first- and second-order state transition matrices of one low-thrust trajectory segment.

Numerical data used for the propagation are given in Table 3 and are mainly taken from Oberle.[37] The motion is two-dimensional and restricted to be in the heliocentric plane. The satellite starts in a circular orbit with the position and velocity of the Earth. The variables to be integrated are then the position and velocity states (four polar coordinate variables), the satellite mass (1 variable) and the control variables (two variables). The trajectory propagation is therefore a seven-dimensional integration.

Table 3. Data of the trajectory propagation

Parameter	Value
Gravitational Parameter	$1.3267\ 10^{20} m^3 s^{-2}$
Initial radius	$1.496\ 10^{11}$ m
Initial velocity	$2.9783\ 10^{4}$ m/s
Initial mass	680 kg
Thrust Magnitude	0.56493 N
Thrust Direction	0^o
I_{sp}	5643 s
Time of Flight	6 days

The standard analytical method integrates directly the state transition matrices from analytical derivatives of the equations of motion.[35] This results in a very large system to integrate with $7 + 7 * 7 + 7 * 7 * 7 = 399$

dimensions. Symmetry and sparsity patterns can be exploited to reduce this number to at most $5 + (7 * 5) + 5 * (7 * 7 + 7)/2 = 180$ dimensions (noting that the control is static). Because such improvements are tedious to implement in the analytic integration of the STMs, the numbers in Table 4 reflect the dense and straightforward implementation with 399 terms. For the multicomplex step differentiation and finite differences, the numerical STM partial derivatives are computed by integrating the original 7-dimensional propagation problem several times for different perturbation steps. In these cases, unlike the analytic case, the sparsity and symmetry patterns of the STMs are easily implemented, and we emphasize that the associated benefit is reflected in the compute times presented in Table 4. For all methods a standard 7^{th}-order Runge-Kutta integrator is used to generate the results. Relative and absolute integration tolerances are set to 10^{-13} for maximum accuracy. Step sizes of $h = 10^{-40}$ and $h = 10^{-4}$ are taken for the multicomplex step and finite difference methods respectively. For finite differences, this step size is obtained after several trial-and-errors to find the best accuracy (this trial and error effort is not included in the speed results). Standard central finite difference formulas are used in the calculations. The reported times and max relative differences reflect the calculation of the full first and second order STMs. The sample spherical component is the term $\frac{\partial^2 r_f}{\partial^2 r_0}$.

Table 4. State transition matrix example for low-thrust spacecraft trajectory.

Method	Sample 2^{nd}-order STM Component	Max. relative difference	Relative computational time
Analytical	$-2.092290564266828\ 10^{-2}$	0.0	1.0 *
MultiComplex Step	$-2.09229056426682\underline{9}\ 10^{-2}$	$5.3\ 10^{-15}$	1.7
AD02	$-2.0922905642668\underline{33}\ 10^{-2}$	$4.0\ 10^{-14}$	4.4
TAPENADE	$-2.09229056426682\underline{6}\ 10^{-2}$	$3.7\ 10^{-14}$	2.1
Finite Differences	$-2.092290\underline{785071782}\ 10^{-2}$	$2.8\ 10^{-6}$	4.5

We can see that the analytical method is again the fastest, but by a much smaller margin than the previous example. This is explained by the fact that a large coupled system has to be integrated. The multicomplex approach is still accurate to machine precision with only a very slight computational handicap. Considering the effort needed to implement the analytical approach, the competitiveness of our approach becomes evident. By comparison, both AD tools, AD02 and TAPENADE, are slower than the MCX approach. Also note that minor changes in the code were required to use AD02 and TAPENADE as some matrix operations (like the *matmul* function) are not supported by these tools. Finally, Finite Difference is clearly the least attractive approach. Its accuracy is poor and it is the slowest of all methods.

In the previous two real-world applications, we find the MCX approach faster than that of AD02 and TAPENADE. However the improvements vary from 260% to 740% for AD02, and from 20% to 30% for TAPENADE, indicating a need to further characterize both applications and other implementation strategies and tools.

CONCLUSION

Many applications in scientific computing require higher order derivatives. This article describes a promising approach to compute higher order derivatives using multicomplex numbers. The theoretical foundation and the basic formulation of this new multicomplex step differentiation method is rigorously introduced. This method is a natural extension to the complex step method recently introduced and now in wide use. The main contribution here is the extension of the complex step derivative to arbitrary order while maintaining the machine precision accuracy that makes both the complex step (for first-order derivatives) and automatic differentiation so attractive. The main results of the paper are formula giving general partial derivatives in terms of imaginary coefficients of multicomplex function evaluations. The main advantage of these expressions is

*this time can likely be reduced by half if the aforementioned symmetries are considered.

that they entail no subtractive cancellation error, and therefore the truncation error can be made arbitrarily (to machine precision) small.

In addition, an efficient implementation strategy using operator and function overloading is outlined. The particular representation of multicomplex numbers which shares the same formal structure as complex numbers makes this overloading particularly simple. The implementation is tested with a simple benchmark function as well as two real-world numerical examples using complicated function calls. The resulting derivative estimates are validated by comparing them to results obtained by other known methods. In both cases of the complicated function calls, the multicomplex method is found to outperform both automatic differentiation and finite differences.

In summary, the multicomplex step method provides a complete differentiation system capable of generating exact high-order partial derivative models for arbitrarily complex systems. This technique combines the accuracy of automatic differentiation with the ease of implementation of finite differences, while being less computationally intensive than either method. We also find that the multicomplex approach is characterized by a shorter development time than that of automatic differentiation, as the theory and code development of the multicomplex technique described in this paper required only a few months to implement. Considering all these advantages the multicomplex method is therefore expected to have a broad potential use.

Future work will apply the multicomplex method to various optimization techniques, such as the second-order Newton's method, where Jacobian and Hessian information is needed. In order to further automate the implementation, the next step is to develop a script that generates automatically the required changes in a code to make it compatible with multicomplex numbers (variable type declarations, 'use' statements, etc). Finally, we intend to exploit the inherent parallelism of the multicomplex step method to further reduce the associated computational cost.

ACKNOWLEDGMENT

This work has been sponsored in part by Thales Alenia Space. We would like to thank Dr. Joris Olympio for his constructive comments on the paper. We are also grateful to Laurent Hascoet for his guidance and suggestions on the TAPENADE implementation.

APPENDIX 1

We give here the proof of Theorem 1. Let $z = z_1 + z_2 i_n$ be an element in $\mathbb{C}^n$. First, by quickly extending the proof of theorem 28.2 in the book of Price,[24] we can say that the set of $2x2$ multicomplex matrices of order $n-1$ of the form

$$M(z) = \begin{pmatrix} z_1 & -z_2 \\ z_2 & z_1 \end{pmatrix} = z_1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + z_2 \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{26}$$

is an isomorphism. In fact, clearly the 0 and identity matrix are of this form. Also the sum and difference of matrices are of this form as well. Regarding the product of matrices, we can readily see that

$$\begin{pmatrix} z_1 & -z_2 \\ z_2 & z_1 \end{pmatrix} \begin{pmatrix} w_1 & -w_2 \\ w_2 & w_1 \end{pmatrix} = \begin{pmatrix} z_1 w_1 - z_2 w_2 & z_1 w_2 + z_2 w_1 \\ -(z_1 w_2 + z_2 w_1) & z_1 w_1 - z_2 w_2 \end{pmatrix} \tag{27}$$

which is also of this form.

Next, the result of the theorem can be deduced by recurrence: z_1 is equivalent to $\begin{pmatrix} z_{11} & -z_{12} \\ z_{12} & z_{11} \end{pmatrix}$. In the same way, z_2 is equivalent to $\begin{pmatrix} z_{21} & -z_{22} \\ z_{22} & z_{21} \end{pmatrix}$. Incorporating these isomorphisms into Eq. 26, we can say that

z is equivalent to $\begin{pmatrix} z_{11} & -z_{12} & -z_{21} & z_{22} \\ z_{12} & z_{11} & -z_{22} & -z_{21} \\ z_{21} & -z_{22} & z_{11} & -z_{12} \\ z_{22} & z_{21} & z_{12} & z_{11} \end{pmatrix}$.

The theorem is then proven by repeating the same operation until a real matrix is recovered. Note that by stopping one step before, we can represent multicomplex numbers by complex matrices as well.

APPENDIX 2

Let the sets

$$\begin{aligned} D_{k,1} &= (z_1 + z_2 i_n)e_{k,1}/(z_1 + z_2 i_n) \in \mathbb{C}^n \\ D_{k,2} &= (z_1 + z_2 i_n)e_{k,2}/(z_1 + z_2 i_n) \in \mathbb{C}^n \end{aligned} \tag{28}$$

where $e_{k,1} = \frac{1+i_k i_{k+1}}{2}$, $e_{k,2} = \frac{1-i_k i_{k+1}}{2}$ for $k = 1, ..., n-1$ are idempotents elements in $\mathbb{C}^n$.

From Price,[24] we can state that two elements in $\mathbb{C}^n$ are divisors of zero if and only if one is $D_{k,1} - 0$ and the other is in $D_{k,2} - 0$ for any $k = 1, ..., n-1$.

REFERENCES

[1] R. Fletcher, *Practical Methods of Optimization*. Wiley, 2nd ed., 2000.

[2] G. Boole, *A Treatise on the Calculus of Finite Differences*. Dover, 2nd ed., 1960.

[3] B. Fornberg, "Numerical Differentiation of Analytic Functions," *ACM Transactions on Mathematical Software*, Vol. 7, No. 4, 1981, pp. 512–526.

[4] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA.: SIAM, 2000.

[5] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland, "ADIFOR - Generating Derivative Codes from Fortran Programs," *Scientific Programming*, Vol. 1, Dec. 1991, pp. 1–29.

[6] V. Pascual and L. Hascoet, "Extension of TAPENADE toward Fortran 95," *Automatic Differentiation: Applications, Theory, and Implementations*, Lecture Notes in Computational Science and Engineering, pp. 171–179, Springer, 2005.

[7] J. D. Pryce and J. K. Reid, "A Fortran 90 code for automatic differentiation," report ral-tr-1998-057, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, 1998.

[8] D. Shiriaev, "ADOL-F: Automatic Differentiation of Fortran Codes," *Computational Differentiation: Techniques, Applications, and Tools*, pp. 375–384, Philadelphia, PA: SIAM, 1996.

[9] J. D. Turner, "Automated generation of high-order partial derivative models," *AIAA Journal*, Vol. 41, Aug. 2003, pp. 1590–1598.

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

[11] J. N. Lyness, "Differentiation Formulas for Analytic Functions," *Mathematics of Computation*, Vol. 22, Apr. 1968, pp. 352–362.

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

[13] J. R. Martins, P. Sturdza, and J. J. Alonso, "The complex-step derivative approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262.

[14] V. N. Vatsa, "Computation of sensitivity derivatives of NavierStokes equations using complex variables," *Advances in Engineering Software*, Vol. 31, Aug. 2000, pp. 655–659.

[15] T. Dargent, "Automatic Minimum Principle Formulation for Low Thrust Optimal Control in Orbit Transfers using Complex Numbers," Sept. 2009. International symposium on space flights dynamics, Toulouse, France.

[16] C. O. Burg and J. C. Newman, "Efficient Numerical Design Optimization Using Highly Accurate Derivatives via the Complex Taylor's Series Expansion Method," *Computers and Fluids*, Vol. 32, Mar. 2003, pp. 373–383.

[17] J. Kim, D. G. Bates, and I. Postlethwaite, "Nonlinear robust performance analysis using complex-step gradient approximation," *Automatica*, Vol. 42, Jan. 2006, pp. 177–182.

[18] J. R. Martins, P. Sturdza, and J. J. Alonso, "The connection between the complex-step derivative approximation and algorithmic differentiation," AIAA Paper 2001-0921, Jan. 2001. AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV.

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

[20] A. A. Abokhodair, "Complex differentiation tools for geophysical inversion," *Geophysics*, Vol. 74, No. 2, 2009, pp. 1–11.

[21] J. D. Turner, "Quaternion-Based Partial Derivative and State Transition Matrix Calculations for Design Optimization," AIAA Paper A02-13810, Jan. 2002. 40th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV.

[22] W. R. Hamilton, "On Quaternions; or on a new System of Imaginaries in Algebra," *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, Vol. 15, 1844, pp. 489–495.

[23] C. Segre, "Le rappresentazioni reali delle forme complesse e gli enti iperalgebrici," *Mathematische Annalen*, Vol. 40, No. 3, 1892, pp. 413–467.

[24] G. B. Price, *An Introduction to Multicomplex Spaces and Functions*. New York: Marcel Dekker Inc., 1991.

[25] N. Fleury, M. R. Detraubenberg, and R. M. Yamaleev, "Commutative Extended Complex Numbers and Connected Trigonometry," *Journal of Mathematical Analysis and Applications*, Vol. 180, Dec. 1993, pp. 431–457.

[26] G. Birkhoff and S. M. Lane, *A Survey of Modern Algebra*. New York: Macmillan, revised edition ed., 1953. p.472.

[27] T. Crilly, "An Argand Diagram for Two by Two Matrices," *The Mathematical Gazette*, Vol. 87, July 2003, pp. 209–216.

[28] D. Rochon, "A generalized Mandelbrot set for bicomplex numbers," *Fractal*, Vol. 8, No. 4, 2000, pp. 355–368.

[29] D. Rochon and S. Tremblay, "Bicomplex Quantum Mechanics: I. The Generalized Schrdinger Equation," *Advances in Applied Clifford Algebras*, Vol. 14, Oct. 2004, pp. 231–248.

[30] J. D. Turner, "An Object-Oriented Operator-Overloaded Quaternion Toolbox," AIAA 20066160, Aug. 2006. AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Keystone, CO.

[31] B. D. Tapley, B. E. Schutz, and G. H. Born, *Statistical Orbit Determination*. Burlington, MA: Elsevier Academic Press, 2004. Sec. 2.3.

[32] F. G. Lemoine, D. E. Smith, M. T. Zuber, G. A. Neumann, and D. D. Rowlands, "A 70th degree lunar gravity model (GLGM-2) from Clementine and other tracking data," *Journal of Geophyical Research*, Vol. 102, No. 16, 1997, p. 339359.

[33] R. H. Battin, *An Introduction to the Mathematics and Methods of Astrodynamics*. AIAA Education Series, Reston, Virginia: American Institute of Aeronautics and Astronautics, revised edition ed., 1999.

[34] O. v. Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, Vol. 37, Dec. 1992, pp. 357–373.

[35] M. Majji, J. D. Turner, and J. L. Junkins, "High Order Methods for Estimation of Dynamic Systems Part 1: Theory," AAS - AIAA Spaceflight Mechanics Meeting, Galveston, TX. To be published in Advances in Astronautical Sciences, 2008.

[36] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193–207.

[37] H. J. Oberle and K. Taubert, "Existence and Multiple Solutions of the Minimum-Fuel Orbit Transfer Problem," *Journal of Optimization Theory and Applications*, Vol. 95, Nov. 1997, pp. 243–262.