Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

Higher Order Approximations for Derivatives using Hypercomplex-Steps

H. M. Nasir

***Abstract*—Complex-step differentiation is a recent popular method to compute a real valued function and its first derivative approximately with second order error using imaginary step size. We propose a generalization of complex-step method to compute a complex valued function and its derivatives up to order $n-1$ with approximate error of order n, for any desired integer n. For this, we use a hypercomplex number system of dimension n and Taylor series expansion of the function at a hypercomplex number. Computations can be performed efficiently by using fast Fourier transform.**

***Keywords*—complex-step differentiation, hypercomplex numbers, automatic differentiation, algorithmic differentiation**

I. Introduction

Computation of derivatives of differentiable functions has many applications in engineering, physics, economics etc. A simple immediate example in economics is the marginal cost and revenues of products. The second order Newton-Raphson root finding algorithm also requires the computation of a function and its first derivative at every point of the successive approximate roots. Higher order convergence requires higher order derivatives of the function to be rooted [1].

A classical method that gives accurate results for derivatives is known as the continuous method in which the function, for which the derivative is sought, is differentiated by hand and the derivatives at the required point are computed by direct substitutions. Although this method gives highly accurate results, obviously, for functions with complicated expressions, this can be a tedious, error-prone and time consuming process. Numerical methods are, therefore, preferred in general.

Among the many methods for numerically approximating derivatives, finite difference method (FDM) is well known. Despite its popularity, FDM is known for its poor accuracy. First, the approximations for derivatives with relatively simple formulas give lower order accuracy, usually one or two and one needs, therefore, to choose the step size very small for better approximation results[2]. Second, in doing so, one faces the 'step size dilemma'. That is, as the step size decreases beyond a bound, the accuracy also practically diminishes as opposed to the expected theoretical improvement [2]. This degrading accuracy occurs due to subtractive cancellations that occur when two very close numbers are differenced, which is often the case for finite difference approximation of derivatives.

Another interesting approach for computing derivatives is the automatic differentiation (AD), also known as algorithmic differentiation or computational differentiation. In this method, the candidate function is transformed into a sequence of elementary operations of intermediate variables and intrinsic functions. Then, the variables are transformed into vectors consisting of the variables and their derivatives up to a desired order. The algebraic operations on these vectors are performed with the use of chain rules of differentiation [3]. This method gives exact results in theoretical considerations and machine precision accuracy in practical computations as there is no truncation error involved[4]. However, there are issues expressed related to the implementation of this method. Transformation of the function expression in to a sequence of elementary operations with simple functions and their derivatives is one such issue that requires user interference for redefining codes [5]. Some reformulations have been considered to address these issues [5,6]. Still, higher order derivatives at a point involve $O(d^2)$ computations, where d is the highest order of derivative desired [5—7].

In [8,9], several methods have been introduced to compute approximate derivatives up to order $n-1$, for a given n, based on approximating the Cauchy integral formula for a complex valued analytic function. Later, in [10], an alternative algorithm approximating the Cauchy integral formula by trapezoidal rule was developed that admits fast Fourier transform (FFT) to efficiently compute the function and its first $n-1$ derivatives for a given n.

Squire and Trapp[11] introduced a method to compute the first derivative of a real valued function with second-order error. In this method, an imaginary perturbation step is introduced to the independent variable and Taylor series expansion is used to obtain an approximation for the first derivative. This method is now popularly known as the complex-step (C-S) differentiation [12]. The C-S method requires only one function evaluation at a complex point and does not suffer from the subtractive difference faced by FDM for small step size. Accuracy is, therefore, increased by comfortably choosing the step size h very small. The order of derivatives obtained by C-S method is limited to one only. Higher order derivatives require a form of finite differences with function evaluations at different complex steps [12]. The C-S method has gained a notable attention and some variations have been considered [13—15].

In this paper, we propose an approximation method to compute a given differentiable complex valued function $f(z)$ and its first $n-1$ derivatives $f^{(k)}(z),\ k=0,1,2,\cdots,n-1$

Department of Mathematics and Statistics
Sultan Qaboos University
Oman

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

simultaneously for any given integer *n* with approximation error of order *n* for all derivatives without using any finite differences. For this, we define a *n*-dimensional hypercomplex number system and use a hypercomplex imaginary step *h* to compute functions of hypercomplex variables efficiently using Discrete Fourier Transforms(DFT) and/or FFT.

Our algorithm establishes connections between the complex-step differentiation and the algorithm in [10] for differentiation of analytic function in the sense that ours is an extension of the former and a reformulation of the latter establishing higher order accuracy.

The rest of the paper is organized as follows. Hypercomplex number systems are described in Section II. The main result of hypercomplex-step differentiation method is introduced in Section III. A DFT/FFT based algorithm of the proposed method is given in Section IV. Numerical results are presented and discussed in Section V with conclusions in Section VI.

II. Commutative Algebra Systems

In this section, we describe the necessary theory of hypercomplex number systems required for the main theme of this paper. A hypercomplex number system is an algebra (a ring with unity) with one or more indeterminates. Examples of hypercomplex systems include complex number systems and non-commutative systems such as quaternion, octonion and Clifford algebras. The indeterminates $\boldsymbol{i}$ in these examples satisfy the quadratic relations $\boldsymbol{i}^2 = \pm 1$. In our work, we consider a generalized hypercomplex system where the indeterminates satisfy a more general condition and preserve commutative property.

For brevity of formulas, we set $N_n = \{0,1,2,\ldots,n-1\}$.

Definition A. *We call a commutative algebra with unit* $\boldsymbol{1}$ *over the complex field an n-hypercomplex system, denoted by* $\mathbb{C}(n,1)$*, if it has only one fundamental unit* $\boldsymbol{e}$ *satisfying* $\boldsymbol{e}^n = 1, \boldsymbol{e} \neq \boldsymbol{1}$. *The elements in* $\mathbb{C}(n,1)$ *are expressed as polynomials in* $\boldsymbol{e}$*, and for convenience, we write them in vector form* $\boldsymbol{a} = \sum_{j=0}^{n-1} a_j \boldsymbol{e}^j = [a_0, a_1, \cdots, a_{n-1}]^t =: [a_j]$.

The fundamental operations on $\mathbb{C}(n,1)$ are given by $\boldsymbol{a} + \boldsymbol{b} = [a_j + b_j]$, $\lambda \boldsymbol{a} = [\lambda a_j]$ and $\boldsymbol{ab} = [\sum_{k=0}^{n-1} a_k b_{(j-k)}]$, where $\lambda \in \mathbb{C}$ and the summation is the cyclic convolution of the components of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with the convention that $b_{(j)} = b_{j \bmod n}$.

The *n*-hypercomplex system is not a division algebra since it has zero divisors. That is, there are non-zero *n*-hypercomplex numbers $\boldsymbol{a}, \boldsymbol{b}$ such that $\boldsymbol{ab} = 0$. For a discussion on zero divisors on generalized hypercomplex systems, see [16].

A. Canonical Representation

An *n*-hypercomplex number in $\mathbb{C}(n,1)$ can be expressed in a form with another set of indeterminate symbols that allow easy manipulations and efficient computations of *n*-hypercomplex expressions and functions.

Proposition A. *For an n-hypercomplex system* $\mathbb{C}(n,1)$ *with indeterminate* $\boldsymbol{e}$ *having fundamental basis* $\{\boldsymbol{1}, \boldsymbol{e}, \boldsymbol{e}^2, \cdots, \boldsymbol{e}^{n-1}\}$*, there is a basis* $\{\hat{\boldsymbol{e}}_0, \hat{\boldsymbol{e}}_1, \hat{\boldsymbol{e}}_2, \cdots, \hat{\boldsymbol{e}}_{n-1}\}$ *with the orthogonal property* $\hat{\boldsymbol{e}}_i \hat{\boldsymbol{e}}_j = 0$ *if* $i \neq j$ *and* $\hat{\boldsymbol{e}}_i^2 = 1$ *for any i, j.*

proof: The multiplication of two *n*-hypercomplex numbers $\boldsymbol{c}, \boldsymbol{a}$ is given by $\boldsymbol{ca} = \boldsymbol{Ca}$, where $\boldsymbol{c} = c_0 + c_1 \boldsymbol{e} + c_2 \boldsymbol{e}^2 + \cdots + c_{n-1} \boldsymbol{e}^{n-1}$ has the circulant matrix representation given by $\boldsymbol{C}(i,j) = c_{(i-j)}$. The eigenpairs of the matrix are

$$\hat{c}_m = \sum_{k=0}^{n-1} c_k w_n^{-km}, m \in N_n, \tag{1}$$

and $\hat{\boldsymbol{e}}_m = \left[1, w_n^m, w_n^{2m}, \cdots, w_n^{(n-1)m}\right]^t$, $m \in N_n$, where $w_n = e^{2i\pi/n}$, $i = \sqrt{-1}$.

If we denote the matrix of eigenvectors $\boldsymbol{W}_n = \frac{1}{\sqrt{n}}[\hat{\boldsymbol{e}}_0, \hat{\boldsymbol{e}}_1, \hat{\boldsymbol{e}}_2, \cdots, \hat{\boldsymbol{e}}_{n-1}]$, we see that $\boldsymbol{W}_n$ is a unitary matrix satisying $\boldsymbol{W}_n^* \boldsymbol{C} \boldsymbol{W}_n = \boldsymbol{D}$ and $\boldsymbol{W}_n^* \boldsymbol{W}_n = \boldsymbol{I}_n$, where the superscript * indicates complex conjugate transpose, $\boldsymbol{D}$ is the diagonal matrix with entries as the eigenvalues $\hat{c}_m$ and $\boldsymbol{I}_n$ is the identity matrix of size *n*. These two relations give the desired property of the new basis.■

Expression (1) is the DFT of $\boldsymbol{c}$ and the inverse DFT is obtained from

$$\boldsymbol{c} = \boldsymbol{C1} = \boldsymbol{W}_n \boldsymbol{D} \boldsymbol{W}_n^* \boldsymbol{1} = \sum_{k=0}^{n-1} \hat{c}_k \hat{\boldsymbol{e}}_k. \tag{2}$$

This gives

$$c_k = \frac{1}{n} \sum_{m=0}^{n-1} \hat{c}_m w_n^{2km}, k \in N_n. \tag{3}$$

Following [17], we call (2) the canonical representation of the *n*-hypercomplex number $\boldsymbol{c} = [c_j]$ and denote the vector of its coefficients as $\hat{\boldsymbol{c}} = (\hat{c}_j)$. We then immediately have

Proposition B. *Let* $\boldsymbol{a} = [a_j]$ *and* $\boldsymbol{b} = [b_j]$ *be two n-hypercomplex numbers with their canonical representation vectors* $\hat{\boldsymbol{a}} = (\hat{a}_j)$ *and* $\hat{\boldsymbol{b}} = (\hat{b}_j)$ *respectively. Then,* $\widehat{\boldsymbol{a} + \boldsymbol{b}} = \hat{\boldsymbol{a}} + \hat{\boldsymbol{b}}$, $\widehat{\lambda \boldsymbol{a}} = \lambda \hat{\boldsymbol{a}}$, $\widehat{\boldsymbol{ab}} = \hat{\boldsymbol{a}} \hat{\boldsymbol{b}}$ *and* $\widehat{\boldsymbol{a}\boldsymbol{b}^{-1}} = \hat{\boldsymbol{a}}/\hat{\boldsymbol{b}}$ *when* $\boldsymbol{b}$ *has an inverse. The operations on the canonical vectors on the right are performed element wise.*

With these basic operations, an *n*-hypercomplex valued function $f(\boldsymbol{z})$ can be expressed in canonical form as $f(\boldsymbol{z}) = \sum_{m=0}^{n-1} f(\hat{z}_m) \hat{\boldsymbol{e}}_m$ and hence we have $\widehat{f(\boldsymbol{z})} = f(\hat{\boldsymbol{z}})$. The coefficients of $f(\boldsymbol{z})$ from the canonical representation vectors can be obtained by an inverse DFT. This process is summarized in

ALGORITHM I: ***Computing*** $f(\boldsymbol{z})$

1. Input $\boldsymbol{z} = [z_0, z_1, \cdots, z_{n-1}]$.
2. Compute $\hat{\boldsymbol{z}} := DFT(z) = (\hat{z}_0, \hat{z}_1, \cdots, \hat{z}_{n-1})$.
3. Compute $f(\hat{\boldsymbol{z}}) := \left(f(\hat{z}_0), f(\hat{z}_1), \cdots, f(\hat{z}_{n-1})\right)$.
4. Compute inverse DFT
   $IDFT(f(\hat{\boldsymbol{z}})) = f(\boldsymbol{z}) =: [f_0, f_1, \cdots, f_{n-1}]$.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

Practically, for large *n*, one can make use of FFT in step 2 and inverse FFT in step 4 for efficient computing of the function $f(\boldsymbol{z})$.

B. *Differentiation*

Assuming Euclidean topology for the system $\mathbb{C}(n,1)$, let $f: U \to \mathbb{C}(n,1)$ be a function on an open set $U \subseteq \mathbb{C}(n,1)$. We state the following definitions and propositions that can be proved analogous to the proofs of similar results on multicomplex functions [18, Section 47].

Definition B. *A function $f: \mathbb{C}(n,1) \to \mathbb{C}(n,1)$ is said to be differentiable at $z_0 \in \mathbb{C}(n,1)$ if the limit*

$$\lim_{\boldsymbol{z}\to\boldsymbol{z}_0}\big(f(\boldsymbol{z})-f(\boldsymbol{z}_0)\big)(\boldsymbol{z}-\boldsymbol{z}_0)^{-1} = f'(\boldsymbol{z}_0)$$

exists. Here, the limit $\boldsymbol{z} \to \boldsymbol{z_0}$ is taken through the z values where $(\boldsymbol{z}-\boldsymbol{z_0})^{-1}$ exist.

Definition C. *A function $f: \mathbb{C}(n,1) \to \mathbb{C}(n,1)$ is said to be holomorphic in an open set $U \subset \mathbb{C}(n,1)$ if it is differentiable for all $z \in U$, and is called analytic if it is holomorphic in entire $\mathbb{C}(n,1)$.*

Proposition C. *A complex holomorphic function can be extended to an n-hypercomplex valued holomorphic function of n-hypercomplex variable.*

Proposition D. *A holomorphic function in $U \subset \mathbb{C}(n,1)$ containing a point $z_0 \in U$ admits Taylor series expansion.*

III. Hypercomplex-Step Differentiation

In this section, we present our main result which we call hypercomplex-step differentiation method. Let $f(z)$ be an analytic function of a real or complex variable *z*. The Taylor series expansion of the extended *n*-hypercomplex valued function $f(\boldsymbol{z})$ at the perturbed point $\boldsymbol{w} = z + h\boldsymbol{e} \in \mathbb{C}(n,1)$ is given by

$$f(z+h\boldsymbol{e}) = f(z) + h\boldsymbol{e}f'(z) + \frac{h^2\boldsymbol{e}^2 f''(z)}{2!} + \cdots + \frac{h^{n-1}\boldsymbol{e}^{n-1}f^{(n-1)}(z)}{(n-1)!} + \frac{h^n\boldsymbol{e}^n f^{(n)}(z)}{n!} + \cdots.$$

This is simplified with the use of $\boldsymbol{e}^n = 1$ as

$$f(z+h\boldsymbol{e}) = \left(f(x) + \frac{h^n f^{(n)}(z)}{n!} + \cdots\right) + h\left(\frac{f'(z)}{1!} + \frac{h^n f^{(n+1)}(z)}{(n+1)!} + \cdots\right)\boldsymbol{e} + \cdots + h^{n-1}\left(\frac{f^{(n-1)}(z)}{(n-1)!} + \frac{h^n f^{(2n-1)}(z)}{(2n-1)!} + \cdots\right)\boldsymbol{e}^{n-1}.$$

Equating the coefficients of $\boldsymbol{e}^k, k \in N_n$, we directly obtain the approximations for $f(z)$ and its derivatives up to order $n-1$ simultaneously as

$$f^{(k)}(z) = \frac{k!}{h^k} Im_k f(z+h\boldsymbol{e}) + O(h^n), k \in N_n, \qquad (4)$$

where Im_k denotes the coefficient of the unit $\boldsymbol{e}^k$.

IV. Implementation

The method described in the previous section reduces the problem of computing the approximation of derivatives at a complex number *z* to an evaluation of the candidate function at a simple *n*-hypercomplex number $\boldsymbol{z} = z + h\boldsymbol{e}$. Computing $f(z+h\boldsymbol{e})$ directly will require a separate implementation to deal with *n*-hypercomplex operations. However, owing to the cyclic nature of the system, evaluation of *n*-hypercomplex valued functions can be efficiently performed via FFT through Algorithm I. Based on this, the following algorithm computes a given function and its first $n-1$ derivatives simultaneously.

ALGORITHM II*: *Hypercomplex-step differentiation

1. Input *z*, *h*, function *f*.
2. Form $\boldsymbol{a} = z + h\boldsymbol{e} =: [z, h, 0, \cdots, 0]$.
3. Compute DFT(a) with $w_n = \mathrm{e}^{2\pi \mathrm{i}/n}$:
   $$\hat{\boldsymbol{a}} = \left(z + hw_n^j\right) =: (\hat{a}_0, \hat{a}_1, \cdots, \hat{a}_{n-1}).$$
4. Compute $f(\hat{\boldsymbol{a}}) = \big(f(\hat{a}_0), f(\hat{a}_1), \cdots, f(\hat{a}_{n-1})\big)$.
5. Compute inverse DFT($f(\hat{\boldsymbol{a}})$)
   $$= f(\boldsymbol{a}) =: [f_0, f_1, \cdots, f_{n-1}]$$
6. Compute derivatives by
   $f^{(k)}(z) = \frac{k!}{h^k} f_k, k \in N_n$.

***Remark*:** Algorithm II is similar to the process given in [19] derived from approximating the coefficients of the Taylor series expansion of complex valued analytic function. The present algorithm was derived from *n*-hypercomplex expansion of an analytic function. This has the advantage of considering other variants of the algorithm that were not possible in the method of [19].

A. *Operation Count*

The DFT in Step 3 of algorithm II is performed in $O(n)$ operations since the *n*-hypercomplex perturbation has only two non-zero coefficients. Function evaluation in Step 4 has $O(n)$ operations. If one needs only the first few derivatives from the output, Step 5 can be reduced to obtain only those derivatives with $O(n)$ operations. Thus over all operations needed to obtain few derivatives with $O(h^n)$ error is only $O(n)$. If all the $n-1$ derivatives are needed, it requires only one inverse FFT in Step 5 with $O(nlog_2 n)$ operations.

Note that the DFT and inverse DFT operations can be reduced by half when *f* is a real valued function of a real variable as half of the DFT, inverse DFT values will be complex conjugates of the other half values [20].

V. Numerical Tests

In order to test the method proposed, the test function used in [8,11,12]

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

$$f(x) = \frac{e^x}{\sin^3 x + \cos^3 x}, x \in \mathbb{R} \qquad (5)$$

is considered. The derivatives of $f(x)$ at $x = 0$ are all integers and the first ten derivatives at $x = 0$ are given in Table I.

The numerical tests were performed on a system having 2.66Ghz speed processor and 2Gb memory with a 32-bit Ubuntu Linux operating system. The algorithms were implemented in Python version 2.7 [21].

A. *Accuracy*

To see the accuracy of the results, the relative errors between the exact derivatives and the approximate derivatives were computed by using $\mathbb{C}(16,1)$ with varying hypercomplex step sizes. The results are given in Table II.

The results show that the relative errors improve towards machine precision when step size *h* decreases and the accuracy of the higher order derivatives are slightly lower than that of the lower order derivatives.

To see the behavior of the errors for various orders with very small steps sizes, the derivatives for the test function (5) were computed using the system with orders $n = 8,16,32$ and 64, and step sizes $h = 2^{-1}, 2^{-2}, \cdots, 2^{-20}$. Fig. 1 shows the relative errors for the first five derivatives including the function itself.

In this test, the relative errors for the derivatives decrease as the step size decreases until the relative error reaches near machine epsilon. After that it increases without bound as the step size further decreases. This phenomenon is prevalent in many approximate differentiation methods for higher derivatives [2]. Nevertheless, higher accuracy is reached when *h* is close to zero for lower order *n* and when *h* is close to one for higher order *n*. This could mean that one can fix a step size *h* and increase the order of the method for improving the accuracy or *vice versa.*

To test this viewpoint, the method for the system $\mathbb{C}(n, 1)$ was tested again by increasing the order of accuracy while keeping the step size *h* fixed. The first ten derivatives of the function in (5) were computed with fixed step sizes $h = 0.5$ and 0.625, and increasing the order of the approximation with $n = 16,17,\cdots,1024$. Fig. 2 shows the relative error of these computations. For clarity, we have restricted the display of order up to 200 only.

The test shows that for any fixed step size *h*, the relative errors can be brought to the machine precision by suitably choosing the order of the method and *vice versa*. This way, our method is comparable to AD in terms of accuracy.

In view of the order $O(h^n)$, we may assume the relation between n and h as $h \approx Eps^{1/n}$ or $n \approx \ln(Eps)/\ln h = \log_2(Eps)/\log_2 h$.

Some of the values for order *n* and corresponding *h* for machine epsilon $Eps = 2^{-52} = 2.220446049250313e^{-16}$ double precision floating point numbers are given in Table III. Fig. 1 and Fig. 2 show that this estimation is agreed in practice.

TABLE I. EXACT DERIVATIVES FOR $f(x)$ AT $x = 0$.

k	$f^{(k)}(0)$
0	1
1	1
2	4
3	4
4	28
5	-164
6	64
7	-13376
8	47248
9	-858224
10	13829824

TABLE II. RELATIVE ERRORS FOR COMPUTED DERIVATIVES USING 16-HYPERCOMPLEX SYSTEM

k	Step Size				
	$h = 2^{-1}$	$h = 2^{-2}$	$h = 2^{-3}$	$h = 2^{-4}$	$h = 2^{-5}$
0	1.8498-04	2.8203-09	4.3077-14	8.6736-19	2.1684-19
1	2.6267-04	4.0051-09	6.1284-14	8.9509-16	1.3426-15
2	1.6181-04	2.4672-09	3.7983-14	2.8039-15	5.3412-15
3	6.0357-04	9.2029-09	1.3446-13	2.4844-13	3.0007-12
4	4.6035-04	7.0193-09	2.8858-13	2.0447-13	5.7168-12
5	4.8001-04	7.3189-09	2.9747-13	3.4020-11	5.7514-10
6	9.5995-03	1.4637-07	2.7371-10	6.9607-09	1.4855-06
7	4.0918-04	6.2389-09	1.1221-11	5.3763-10	4.6193-07
8	1.1659-03	1.7773-08	2.3818-09	3.9658-07	9.4416-05
9	7.4612-04	1.1372-08	2.6594-09	3.2767-07	6.1315-04
10	5.8381-04	8.9155-09	2.4154-08	9.7246-06	3.6090-02

TABLE III. ORDER AND CORRESPONDING STEP SIZE

n	2	4	8	16	32	64	128
h	1.4901×10^{-8}	1.2207×10^{-4}	1.105×10^{-2}	1.051×10^{-1}	0.32421	0.5694	0.7546

B. *Computational Efficiency*

The following tools were chosen for comparison testing: Numdifftools[23] which uses finite difference method for derivatives up to order four with Romberg improvement and Algopy[23] that uses AD. The reasons for choosing these tools are that they are comparable to other existing differentiation tools using methods of same kind in terms of accuracy and efficiency, and all have been implemented in Python so that the comparison made in this test will be platform unbiased.

The CPU time to compute the derivatives up to a required order was obtained by executing the appropriate algorithm 1000 times and averaging. The relative CPU time with respect to computing the raw function (assuming it takes unit time) were obtained by the ratio *RelCPU = CPU(Der)/CPU(fun)* for comparison. The total relative CPU times to compute the first four derivatives by the tools are listed in Table IV.

Table IV also gives the relative CPU time to compute the derivative up to order four using Numdifftools, Algopy and hypercomplex-step methods. This indicates that the CPUtime for Algopy grows substantially as the order increases while hypercomplex-step method performs well. Numdifftools which computes only up to order four takes high CPUtime.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

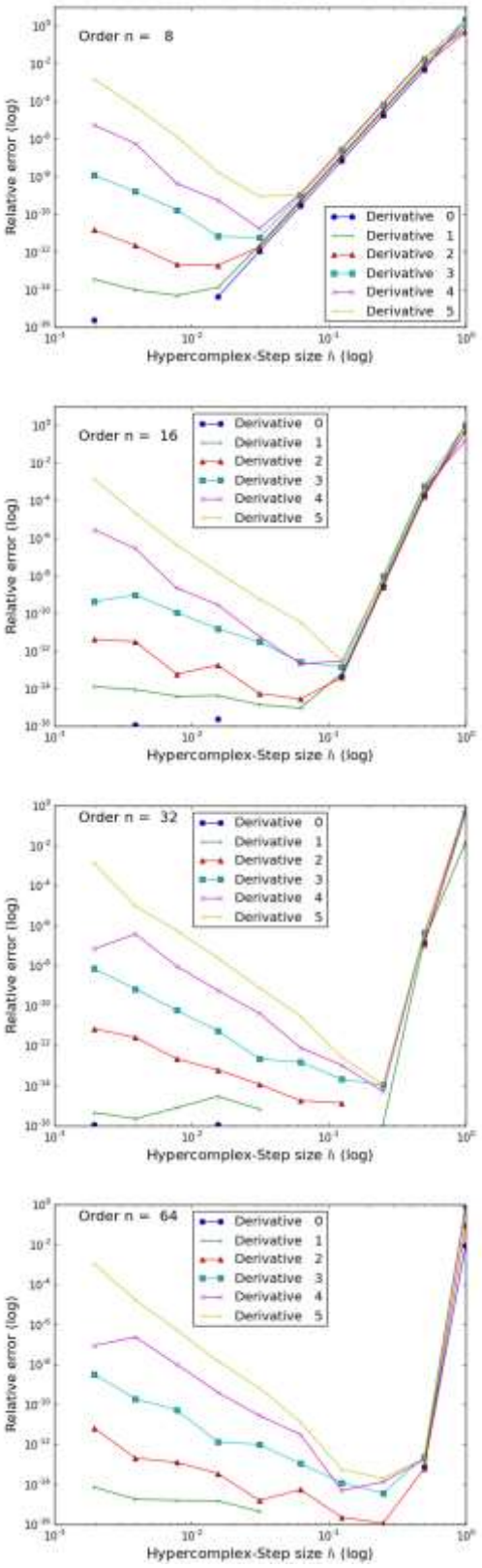

Figure 1. Relative errors for derivatives versus step size *h*

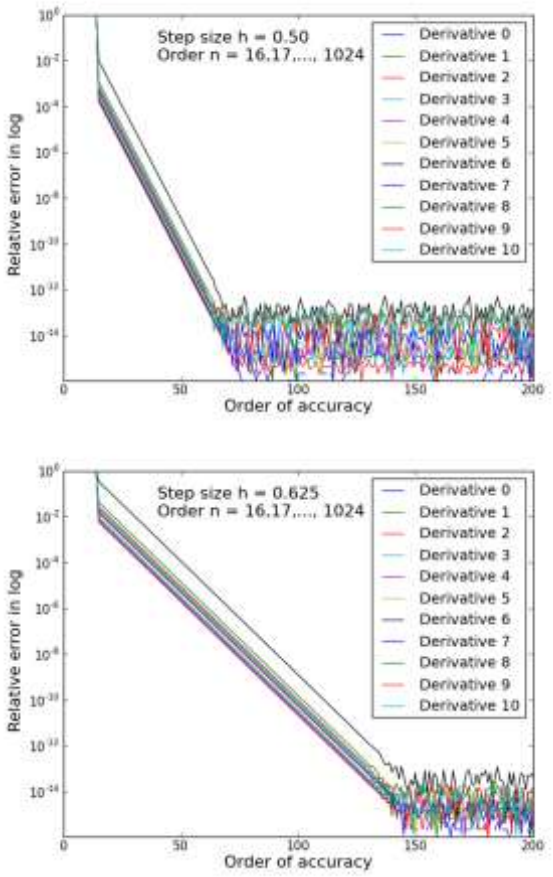

Figure 2. Relative error for fixed step size *h* and varying order *n*.

TABLE IV. ORDER AND CORRESPONDING STEP SIZE

deriv.	NumdiffTools	Algopy	Hypercomplex-step
0	1	1	1
1	119.3	27.4	3.1
2	240.6	45.7	4.3
3	578.2	65.6	5.2
4	718.9	87.7	5.5
99	--	10004.1	66.9

Efficiency for higher order derivatives was tested by comparing the relative CPU times for Algopy and the proposed hypercomplex-step methods for the derivatives of orders up to 100. Fig. 3 shows the plots of the relative CPU times against the order. This test shows that the execution time for Algopy grows quadratically as order increases while for hypercomplex-step method it grows only linearly.

We mention that while Algopy produces exact results, the hypercomplex-step method produces approximate values whose errors can be brought down to near machine precision by suitably choosing the step size *h* for a specified order.

Proc. of The Third Intl. Conf. on Advances in Computing, Electronics and Communication - ACEC 2015

ISBN: 978-1-63248-064-4 doi: 10.15224/ 978-1-63248-064-4-106

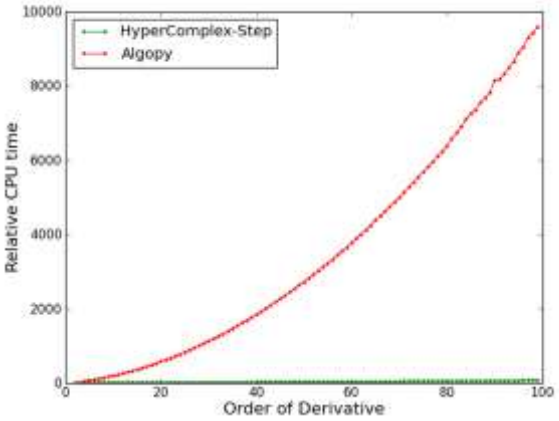

Figure 3. Relative CPU time versus Order

VI. Conclusion

The proposed method computes a given function and its first $n-1$ derivatives simultaneously with approximation error of order n. Accuracy up to near machine precision can be achieved by choosing a step size h corresponding to an order n. The method computes derivatives of real and complex valued functions efficiently with the use of FFT thereby improving the previous methods in computing time with fully automated implementation.

References

[1] A. Householder, The Numerical Treatment of a Single Nonlinear Equation, McGraw-Hill Publishing Co., NewYork, 1970.

[2] M. Ramesh Kumar and G. Uthra, A Study on Numerical Stability of Finite Difference Formulae for Numerical Differentiation and Integration, Annals of Pure and Applied Mathematics, 2014, vol. 8:2, pp. 27-36.

[3] A. Griewank and A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation (2nd Edition) SIAM, 2008.

[4] A. Griewank, J. Utke and A. Walther, Evaluating Higher Order Derivative Tensors by Forward Propagation of Univariate Taylor Series, Mathematics of Computation, vol. 69:231, 2000, pp. 1117-1130.

[5] S. F. Walter and L. Lehman, Algorithmic Differentiation with AlgoPy, J. Comp. Science, vol. 4, 2013, pp. 334-344.

[6] J. A. Fike and J. A. Alonso, Automatic Differentiation through the Use of Hyper-Dual Number for Second Derivative, In Recent Advances in Algorithmic Differentiation. Eds. Forth, S., Hovland, P., Phipps, E., Utke, J. and Walther A., Lecture Notes in Computational Science and Engineering: vol. 87, pp. 163-173, Springer Berlin Heidelberg, 2012.

[7] J. A. Fike, S., Jongsma, J. A. Alonso and E. Van der Weide, Optimization with Gradient and Hessian Information Calculated using Hyper-Dual Numbers, 29th AIAA Applied Aerodynamics Conference, pp. 27-30, Honolulu, Hawaii, June 2011.

[8] J. N. Lyness, Differentiation Formulas for Analytic Functions, Mathematics of Computation.vol. 22, 1968, pp. 352-362.

[9] J. N. Lyness, and C. B. Moler, Numerical Differentiation of Analytic Functions, SIAM journal on Numerical Analysis,vol. 4, 1967, pp. 202-210.

[10] B. Fornberg, Numerical Differentiation of Analytic Functions, ACM Transactions of Mathematical Software, vol. 7:4, 1981, pp. 512-526.

[11] W. Squire and G. Trapp, Using Complex Variables to Estimate Derivatives of Real Functions, SIAM Review, vol. 40:1, 1998, pp. 110-112.

[12] J. R. R. A. Martins, P. Sturda and J. J. Alonso, The Complex-Step Derivative Approximation, ACM Transactions on Mathematical Software, vol. 29:3, 2003, pp. 245-262.

[13] R. Abreu, D. Stich and J. Morales, On the Generalization of the Complex Step Method, J. Comp. Appl. Math., vol. 241, 2013, pp. 84-102.

[14] K. L. Lai, Generalizations of the Complex-Step Derivative Approximation, PhD Thesis, University at Buffalo, Buffalo, NY, Sept. 2006.

[15] R. W. Ibrahim and H. A. Jalab, The Fractional Complex Step Method, Discrete Dynamics in nature and Society, Hindawi Publishers, 2013, doi:10.1155/2013/515973, 2013.

[16] H. M. Nasir, A New Class of Multicomplex Algebra with Applications, Mathematical Sciences International Research Journal, vol. 2:2, 2013, pp. 163-168.

[17] C. M. Davenport, A Commutative Hypercomplex Algebra with Associated Function Theory, In Clifford Algebras With Numeric And Symbolic Computations, Eds. Ablamowicz, R., Lounesto, P., Parra, J.M., Birkhauser Boston Inc., Cambridge, MA, USA, pp. 213–227, 1996.

[18] G. B. Price, Introduction to multicomplex spaces and functions, New York, Marcel Dekker, Inc., 1991.

[19] P. Henrici, Fast Fourier Methods in Computational Complex Analysis, SIAM Rev., vol. 21, 1979, pp. 481-527.

[20] J. S. Walker, Fast Fouerier Transforms, CRC Press, 1996.

[21] G. van Rossum, Python tutorial, Technical Report CS-R9526: Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[22] P. A. Brodtkorb, Numdifftools, http://code.google.com/p/numdifftools/, 2009.

[23] F. W. Sebasteian, ALGOPY:Alogorithmic Differentiation in Python, http://pypi.python.org/pypi/algopy, 2009.

About Author:

Haniffa Mohamed Nasir obtained his B.Sc. degree in mathematics from University of Jaffna, Sri Lanka and obtained his M.Eng. and Ph.D. degrees in numerical analysis of wave scattering problems from The University of Electro-Communications, Tokyo, Japan. He was Lecturer in University of Jaffna, Sri Lanka, Senior Lecturer in University of Peradeniya, Sri Lanka and served as Head of Department since 2010. Currently, he is Assistant Professor in the Department of Mathematics and Statistics, Sultan Qaboos University, Oman.

Dr. Nasir is senior member of International Association of Computer Science and Information Technology (IACSIT), Institute of Engineers and Doctors(IRED), Sri Lanka Association for Advancement of Science (SLAAS), Senior Scientist Forum (SSF), and member of Society of Industrial and Applied Mathematics (SIAM).