Higher order numerical differentiation
on the Infinity Computer

Yaroslav D. Sergeyev

Abstract There exist many applications where it is necessary to approximate numerically derivatives of a function which is given by a computer procedure. In particular, all the fields of optimization have a special interest in such a kind of information. In this paper, a new way to do this is presented for a new kind of a computer – the Infinity Computer – able to work numerically with finite, infinite, and infinitesimal numbers. It is proved that the Infinity Computer is able to calculate values of derivatives of a higher order for a wide class of functions represented by computer procedures. It is shown that the ability to compute derivatives of arbitrary order automatically and accurate to working precision is an intrinsic property of the Infinity Computer related to its way of functioning. Numerical examples illustrating the new concepts and numerical tools are given.

Keywords Higher order numerical differentiation, infinite and infinitesimal numbers, Infinity Computer.

Mathematics Subject Classification (2000) 65K05, 90C26, 90C56

1 Introduction

In many practical applications related to the scientific computing (e.g., in global and local optimization, numerical simulation, approximation, etc.) it is necessary to calculate derivatives of a function $g(x)$ which is given by a computer procedure calculating its approximation $f(x)$. Very often a user working with the computing code $f(x)$ is not the person who has written this code. As a result, for the user the program calculating $y = f(x)$ is just a black box, i.e., if it has as the input a value x then the program returns the corresponding value y and the user does not know the internal structure of the program. As a result, when for solving an applied problem the usage of derivatives is required and a procedure for evaluating the exact value of $f'(x)$ is not available, we face the necessity to approximate $f'(x)$ in a way.

Yaroslav D. Sergeyev
DEIS – University of Calabria, Via P. Bucci 42C, 87036 Rende (CS), Italy
Software Department, N.I. Lobatchevsky State University, Nizhni Novgorod, Russia
E-mail: yaro@si.deis.unical.it

In particular, this situation happens very often in the black box global and local optimization (see [5, 11, 19, 20]) and related application areas. Let us give a simple but important example (see, e.g., [19, 20, 23]) related to the problem of finding the minimal root of an equation $f(x) = 0$ where $x \in [a, b]$ and $f(x)$ is multiextremal (as a result, there can be several different roots over $[a, b]$), given by a computer program and such that $f(a) > 0$. This problem arises in many applications, such as time domain analysis (see [3]), filter theory (see [7]), and wavelet theory (see [21]) and can be interpreted, for instance, as follows.

It is necessary to know the behavior of a device over a time interval $[a, b]$. The device starts to work at the time $x = a$ and it functions correctly while for $x \geq a$ the computer procedure calculating $f(x)$ returns values $f(x) > 0$. Of course, at the initial moment, $x = a$, the device works correctly and $f(a) > 0$. It is necessary either to find an interval $[a, x^*]$ such that

$$f(x^*) = 0, \quad f(x) > 0, \quad x \in [a, x^*), \quad x^* \in (a, b], \tag{1}$$

or to prove that x^* satisfying (1) does not exist in $[a, b]$. Efficient methods proposed recently for solving this problem (see [6, 18, 19]) strongly use ideas developed in the field of global optimization. They require calculating the first derivative $f'(x)$ of $f(x)$ and since a program calculating $f'(x)$ is usually not available, the problem of finding an approximation of $f'(x)$ arises.

There exist several approaches to tackle this problem. First, numerical approximations are used for this purpose (see e.g., [9] for a detailed discussion). In applications, the following three simple formulae (more complex and numerically more expensive approximations can be found in [9]) are often used

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \quad f'(x) \approx \frac{f(x) - f(x-h)}{h}, \tag{2}$$

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{3}$$

by practitioners. However, these procedures are fraught with danger (see [9]) since eventually round-off errors will dominate calculation. As h tends to zero, both $f(x + h)$ and $f(x - h)$ tend to $f(x)$, so that their difference tends to the difference of two almost equal quantities and thus contains fewer and fewer significant digits. Thus, it is meaningless to carry out these computations beyond a certain threshold value of h. Calculations of higher derivatives suffer from the same problems.

The complex step method (see [8]) allows one to improve approximations of $f'(x)$ avoiding subtractive cancellation errors present in (2), (3) by using the following formula to approximate $f'(x)$

$$f'(x) \approx \frac{Im[f(x + ih)]}{h}, \tag{4}$$

where $Im(u)$ is the imaginary part of u. Though this estimate does not involve the dangerous difference operation, it is still an approximation of $f'(x)$ because it depends on the choice of the step h.

Another approach consists of the usage of symbolic (algebraic) computations (see, e.g., [4]) where $f(x)$ is differentiated as an expression in symbolic form in contrast to manipulating of numerical quantities represented by the symbols used to express $f(x)$. Unfortunately this approach can be too slow when it is applied to long codes coming from real world applications.

There exist an extensive literature (see, e.g., [1, 2, 5] and references given therein) dedicated to automatic (algorithmic) differentiation (AD) that is a set of techniques based on

the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. By applying the chain rule of derivation to elementary operations this approach allows one to compute derivatives of arbitrary order automatically with the precision of the code representing $f(x)$.

Implementations of AD can be broadly classified into two categories that have their advantages and disadvantages (see [2, 5] for a detailed discussion): (i) AD tools based on source-to-source transformation changing the semantics by explicitly rewriting the code; (ii) AD tools based on operator overloading using the fact that modern programming languages offer the possibility to redefine the semantics of elementary operators. In particular, the dual numbers extending the real numbers by adjoining one new element d with the property $d^2 = 0$ (i.e., d is nilpotent) can be used for this purpose (see, e.g., [1]). Every dual number has the form $v = a + db$, where a and b are real numbers and v can be represented as the ordered pair (a, b). On the one hand, dual numbers have a clear similarity with complex numbers $z = a + ib$ where $i^2 = -1$. On the other hand, speaking informally it can be said that the imaginary unit d of dual numbers is a close relative to infinitesimals (we mean here a general non formalized idea about infinitesimals) since the square (or any higher power) of d is exactly zero and the square of an infinitesimal is 'almost zero'.

All the methods described above use traditional computers as computational devices and propose a number of techniques to calculate derivatives on them. In this paper, a new way to calculate derivatives numerically is proposed. It is made by using a new kind of a computer – the Infinity Computer – introduced in [13–15] and able to work *numerically* with finite, infinite, and infinitesimal quantities. This computer is based on a new applied point of view on infinite and infinitesimal numbers (that is not related to non-standard analysis) introduced in [12, 14]. The new approach does not use Cantor's ideas and works with infinite and infinitesimal numbers being in accordance with Aristotle's principle 'The part is less than the whole'.

We conclude this introduction by emphasizing that traditional approaches for differentiation considered above have been developed *ad hoc* for solving this problem as additional tools that should be used together with the traditional computers. Without these additional tools the traditional computers are not able to calculate derivatives of functions defined by computer procedures. In this paper, it is shown that the ability to compute derivatives of arbitrary order automatically and accurate to working precision is an intrinsic property of the Infinity Computer related to *its way of functioning*. This is just one of the particular features offered to the user by the Infinity Computer. Naturally, this is a direct consequence of the fact that it can execute numerical computations with infinite and infinitesimal quantities explicitly.

2 Representation of numbers at the Infinity Computer

In [12, 14, 16, 17], a new powerful numeral system has been developed to express finite, infinite, and infinitesimal numbers in a unique framework. The main idea consists of measuring infinite and infinitesimal quantities by different (infinite, finite, and infinitesimal) units of measure. In this section we give just a brief introduction to the new methodology that can be found in a rather comprehensive form in the survey [14] or in the monograph [12] written in a popular manner.

A new infinite unit of measure has been introduced as the number of elements of the set $\mathbb{N}$ of natural numbers. It is expressed by a new numeral ① called *grossone*. It is necessary to emphasize immediately that the infinite number ① is not either Cantor's $\aleph_0$ or ω and the

new approach is not related to the non-standard analysis. For instance, one of the important differences consists of the fact that infinite integer positive numbers that can be viewed by using numerals including grossone can be interpreted in the terms of the number of elements of certain infinite sets. Another difference consists of the fact that ① has both cardinal and ordinal properties as usual finite natural numbers.

Formally, grossone is introduced as a new number by describing its properties postulated by the *Infinite Unit Axiom* (IUA) (see [12, 14]). This axiom is added to axioms for real numbers similarly to addition of the axiom determining zero to axioms of natural numbers when integer numbers are introduced. Inasmuch as it has been postulated that grossone is a number, all other axioms for numbers hold for it, too. Particularly, associative and commutative properties of multiplication and addition, distributive property of multiplication over addition, existence of inverse elements with respect to addition and multiplication hold for grossone as for finite numbers. This means that the following relations hold for grossone, as for any other number

$$0 \cdot ① = ① \cdot 0 = 0, \quad ① - ① = 0, \quad \frac{①}{①} = 1, \quad ①^0 = 1, \quad 1^① = 1, \quad 0^① = 0. \tag{5}$$

To express infinite and infinitesimal numbers at the Infinity Computer, records similar to traditional positional numeral systems can be used (see [12–14]). Numbers expressed in this new positional systems with the radix ① are called hereinafter *grossnumbers*. In order to construct a number C in this system, we subdivide C into groups corresponding to powers of grossone:

$$C = c_{p_m}①^{p_m} + \ldots + c_{p_1}①^{p_1} + c_{p_0}①^{p_0} + c_{p_{-1}}①^{p_{-1}} + \ldots + c_{p_{-k}}①^{p_{-k}}. \tag{6}$$

Then, the record

$$C = c_{p_m}①^{p_m} \ldots c_{p_1}①^{p_1} c_{p_0}①^{p_0} c_{p_{-1}}①^{p_{-1}} \ldots c_{p_{-k}}①^{p_{-k}} \tag{7}$$

represents the number C, where finite numbers $c_i \neq 0$ called *grossdigits* can be positive or negative. They show how many corresponding units should be added or subtracted in order to form the number C. Grossdigits can be expressed by several symbols.

Numbers p_i in (7) called *grosspowers* can be finite, infinite, and infinitesimal, they are sorted in the decreasing order with $p_0 = 0$

$$p_m > p_{m-1} > \ldots > p_1 > p_0 > p_{-1} > \ldots p_{-(k-1)} > p_{-k}.$$

In the record (7), we write $①^{p_i}$ explicitly because in the new numeral positional system the number i in general is not equal to the grosspower p_i (see [14] for a detailed discussion).

Finite numbers in this new numeral system are represented by numerals having only one grosspower $p_0 = 0$. In fact, if we have a number C such that $m = k = 0$ in representation (7), then due to (5), we have $C = c_0①^0 = c_0$. Thus, the number C in this case does not contain grossone and is equal to the grossdigit c_0 being a conventional finite number expressed in a traditional finite numeral system.

The simplest infinitesimal numbers are represented by numerals C having only finite or infinite negative grosspowers, e.g., $6.73①^{-4.7}56.7①^{-150}$. The simplest infinitesimal number is $\frac{1}{①} = ①^{-1}$ being the inverse element with respect to multiplication for ①:

$$①^{-1} \cdot ① = ① \cdot ①^{-1} = 1. \tag{8}$$

Note that all infinitesimals are not equal to zero. Particularly, $\frac{1}{①} > 0$ because it is a result of division of two positive numbers.

In the context of the numerical differentiation discussed in this paper, it is worth mentioning that (without going in a detailed and rather philosophical discussion on the topic 'Can or cannot dual numbers be viewed as a kind of infinitesimals?') there exist two formal differences between infinitesimals C from (7) and dual numbers (see, e.g., [1]). First, for any infinitesimal C it follows $C^2 > 0$ (for instance, $(①^{-1})^2 > 0$) whereas for dual numbers we have $d^2 = 0$. Second, in the context of [1] the element d represented as $(0, 1)$ has not its inverse and infinitesimals C have their inverse.

The simplest infinite numbers are expressed by numerals having positive finite or infinite grosspowers. They have infinite parts and can also have a finite part and infinitesimal ones. For instance, the number

$$1.5①^{14.2}(-10.645)①^5 7.89①^0 81①^{-4.2}72.8①^{-60}$$

has two infinite parts $1.5①^{14.2}$ and $-10.645①^5$ one finite part $7.89①^0$ and two infinitesimal parts $81①^{-4.2}$ and $72.8①^{-60}$. All of the numbers introduce above can be grosspowers, as well, giving so a possibility to have various combinations of quantities and to construct terms having a more complex structure.

A working software simulator of the Infinity Computer has been implemented and the first application – the Infinity Calculator – has been realized. We conclude this section by emphasizing the following important issue: the Infinity Computer works with infinite, finite, and infinitesimal numbers *numerically*, not symbolically (see [15]).

3 Numerical differentiation

Let us return to the problem of numerical differentiation of a function $g(x)$. We suppose that a set of elementary functions $(\sin(x), \cos(x), a^x$ etc.) is represented at the Infinity Computer by one of the usual ways used in traditional computers (see, e.g. [10]) involving the argument x, finite constants, and four arithmetical operations. A programmer writes a program P that should calculate $g(x)$ using the said implementations of elementary functions, the argument x, and finite constants connected by four arithmetical operations. Obviously, P calculates a numerical approximation $f(x)$ of the function $g(x)$. As a rule, the programmer does not use analytical formulae of $f'(x), f''(x), \ldots f^{(k)}(x)$ to write the program calculating $f(x)$. We suppose that $f(x)$ approximates $g(x)$ sufficiently well with respect to some criteria and we shall not discuss the goodness of this approximation in this paper.

Then, as often happens in the scientific computing, a user takes the program P calculating $f(x)$ and is interested to calculate $f'(x)$ and higher derivatives numerically by using this program. Computer programs for calculating $f'(x), f''(x), \ldots f^{(k)}(x)$ and their analytical formulae are unavailable and the internal structure of the program calculating $f(x)$ is unknown to the user.

In this situation, our attention will be attracted to the problem of a numerical calculation of the derivatives $f'(x), f''(x), \ldots f^{(k)}(x)$ and to the information that can be obtained from the computer procedure P calculating $f(x)$ for this purpose when it is executed at the Infinity Computer. The following theorem holds.

Theorem 1 *Suppose that: (i) for a function $f(x)$ calculated by a procedure implemented at the Infinity Computer there exists an unknown Taylor expansion in a finite neighborhood $\delta(y)$ of a finite point y; (ii) $f(x), f'(x), f''(x), \ldots f^{(k)}(x)$ assume finite values or are equal*

to zero for $x \in \delta(y)$; (iii) $f(x)$ has been evaluated at a point $y + ①^{-1} \in \delta(y)$. Then the Infinity Computer returns the result of this evaluation in the positional numeral system with the infinite radix ① in the following form

$$f(y + ①^{-1}) = c_0①^0 c_{-1}①^{-1} c_{-2}①^{-2} \ldots c_{-(k-1)}①^{-(k-1)} c_{-k}①^{-k}, \qquad (9)$$

where

$$f(y) = c_0, \ f'(y) = c_{-1}, \ f''(y) = 2!c_{-2}, \ \ldots \ f^{(k)}(y) = k!c_{-k}. \qquad (10)$$

Proof Due to its rules of operation (see (6), (7)), the Infinity Computer collects different exponents of ① in independent groups $c_{p_{-i}}①^{p-i}$ with finite grossdigits $c_{p_{-i}}$ when it calculates $f(y + ①^{-1})$. Since functions $f(x), f'(x), f''(x), \ldots f^{(k)}(x)$ assume finite values or are equal to zero in $\delta(y)$ which is also finite, the highest grosspower in the number (9) is necessary less or equal to zero. Thus, the number that the Infinity Computer returns can have only a finite and infinitesimal parts.

Four arithmetical operations (see [14, 15]) executed by the Infinity Computer with the operands having finite integer grosspowers in the form (7) produce only results with finite integer grosspowers. This fact ensures that the result $f(y + ①^{-1})$ can have only integer non-positive grosspowers in (9). Due to the rules of the positional system (see (6), (7)), the number $f(y + ①^{-1})$ from (9) can be written as follows

$$f(y + ①^{-1}) = c_0①^0 c_{-1}①^{-1} c_{-2}①^{-2} \ldots c_{-(k-1)}①^{-(k-1)} c_{-k}①^{-k} =$$

$$c_0①^0 + c_{-1}①^{-1} + c_{-2}①^{-2} + \ldots + c_{-(k-1)}①^{-(k-1)} + c_{-k}①^{-k}. \qquad (11)$$

The Infinity Computer while calculates the value $f(y + ①^{-1})$ does not use the Taylor expansion for $f(x)$, it just executes commands of the program. However, this unknown Taylor expansion for $f(x)$ (we emphasize that it is unknown for: the Infinity Computer itself, for the programmer, and for the user) exists in the neighborhood $\delta(y)$ of the point y, for a point $x = y + h \in \delta(y)$, $h > 0$. Thus, it should be true

$$f(y + h) = f(y) + f'(y)h + f''(y)\frac{h^2}{2} + \ldots + f^{(k)}(y)\frac{h^k}{k!} + \ldots \qquad (12)$$

By assuming $h = ①^{-1}$ in (12) and by using the fact that $①^0 = 1$ (see (5)) we obtain

$$f(y + ①^{-1}) = f(y)①^0 + f'(y)①^{-1} + \frac{f''(y)}{2}①^{-2} + \ldots + \frac{f^{(k)}(y)}{k!}①^{-k} + \ldots \qquad (13)$$

The uniqueness of the Taylor expansion allows us to obtain (9) by equating the first $k + 1$ coefficients of ① in (13) with grossdigits $c_0, c_{-1}, c_{-2}, \ldots c_{-(k-1)}, c_{-k}$ in (11) completing so the proof. $\qquad \square$

Let us comment upon the theorem. It describes a situation where a user needs to evaluate $f(x)$ and its derivatives at a point $x = y$ but analytic expressions of $f(x), f'(x), f''(x), \ldots f^{(k)}(x)$ are unknown and computer procedures for calculating $f'(x), f''(x), \ldots f^{(k)}(x)$ are unavailable. Moreover, the internal structure of the procedure P calculating $f(x)$ can also be unknown to the user. In this situation, instead of the usage of, for instance, traditional formulae (2), (3) for an approximation of $f'(x)$, the user evaluates $f(x)$ at the point $x = y + ①^{-1}$ at the Infinity Computer. Note that if P has been written by the programmer for the Infinity Computer, then the user just runs P without any intervention on the code of P. In the case when P has been written for traditional computers, in order to transfer it to the Infinity

Computer, variables and constants used in P should be just redeclared as grossnumbers (7). Traditional arithmetic operations are then overloaded due to the rules defined in [14,15].

The operation of evaluation of $f(x)$ at the point $x = y + ①^{-1}$ returns a number in the form (9) from where the user can easily obtain values of $f(y)$ and $f'(y), f''(y), \dots f^{(k)}(y)$ as shown in (10) without any knowledge of the Taylor expansion of $f(x)$ and of the analytic formulae and computer procedures for evaluating derivatives. Due to the fact that the Infinity Computer is able to work with infinite and infinitesimal numbers numerically, the values $f'(y), \dots f^{(k)}(y)$ are calculated exactly at the point $x = y$ without introduction of dangerous operations (2), (3) (or (4)) related to the necessity to use finite values of h when one works with traditional computers. We emphasize also that the user obtains the function value and the values of the derivatives after calculation of $f(x)$ at a single point.

It is worthy to notice that numerical operations that the Infinity Computer performs when it executes the program $f(x)$ can be viewed as an automatic rewriting of $f(x)$ from the basis in x into the basis in ① by setting $x = y + ①^{-1}$ with y being a finite number. The numerical finite value of y is then combined with other finite numbers present in the program and they all are collected as finite coefficients (i.e., grossdigits) of grosspowers of ①. In some sense this is similar to rearrangements that often are executed when one works with wavelets (see [21]) or with formal power series (see [22]).

Let us consider some numerical examples. Their results can be checked by the reader directly on systems using symbolic calculations (e.g., MAPLE) by taking instead of $①^{-1}$ a symbolic parameter, let say, a, thinking about a as an infinitesimal number and by calculating then $f(y + a)$ where y is a number. The crucial difference of the Infinity Computer with respect to systems executing symbolic computations consists of the fact that the Infinity Computer works with infinite, finite, and infinitesimal numbers *numerically*, not symbolically. Naturally, this feature of the Infinity Computer becomes very advantageous when one should execute complex numerical computations.

Example 1 Suppose that we have a computer procedure implementing the following function $g(x) = x^3$ as $f(x) = x \cdot x \cdot x$ and we want to evaluate the values $f(y), f'(y), f''(y)$, and $f^{(3)}(y)$ at the point $y = 5$. The Infinity Computer executes the following operations

$$f(5 + ①^{-1}) = 5①^0 1①^{-1} \cdot 5①^0 1①^{-1} \cdot 5①^0 1①^{-1} =$$

$$25①^0 10①^{-1} 1①^{-2} \cdot 5①^0 1①^{-1} = 125①^0 75①^{-1} 15①^{-2} 1①^{-3}. \qquad (14)$$

From (14), by applying (10) we obtain that

$$f(5) = 125, \quad f'(5) = 75, \quad f''(5) = 2! \cdot 15 = 30, \quad f^{(3)}(5) = 3! \cdot 1 = 6,$$

that are correct values of $f(x)$ and the derivatives at the point $y = 5$.

Let us check this numerical result analytically by taking a generic point y. Then we obtain

$$f(y + ①^{-1}) = (y + ①^{-1})^3 = (y + ①^{-1}) \cdot (y + ①^{-1}) \cdot (y + ①^{-1}) = \qquad (15)$$

$$y^3 + 3y^2 ①^{-1} + 3y①^{-2} + ①^{-3} = y^3①^0 3y^2①^{-1} 3y①^{-2} 1①^{-3}. \qquad (16)$$

By applying (10) we have the required values

$$f(y) = y^3, \quad f'(y) = 3y^2, \quad f''(y) = 2! \cdot 3y = 6y, \quad f^{(3)}(y) = 3! \cdot 1 = 6.$$

That coincide with the respective analytical derivatives calculated at the point $x = y$:

$$f'(x) = 3x^2, \quad f''(x) = 6x, \quad f^{(3)}(x) = 6. \qquad \square$$

Example 2 Suppose that we have the following function $g(x) = x + \sin(x)$ and it is represented in the Infinity Computer as

$$f(x) = x + \widetilde{\sin}(x), \tag{17}$$

where $\widetilde{\sin}(x)$ is a computer implementation of $\sin(x)$. If we want to evaluate $f(x), f'(x), f''(x)$, and $f^{(3)}(x)$ at a point y, by taking $k = 3$ in (9) we obtain

$$f(y + ①^{-1}) = (y + \widetilde{\sin}(y))①^0(1 + \widetilde{\sin}'(y))①^{-1}\frac{\widetilde{\sin}''(y)}{2}①^{-2}\frac{\widetilde{\sin}^{(3)}(y)}{3!}①^{-3},$$

where the result depends on the way of implementation of $\widetilde{\sin}(x)$. For example, suppose for the illustrative purpose that in the neighborhood of the point $y = 0$ the Infinity Computer uses the following simple implementation

$$\widetilde{\sin}(x) = x - \frac{x \cdot x \cdot x}{6}$$

being the first two items in the corresponding Taylor expansion. Then the computer program $f(x)$ becomes

$$f(x) = x + x - \frac{x \cdot x \cdot x}{6}$$

and the Infinity Computer with $y = 0$ works as follows

$$f(0 + ①^{-1}) = 0 + ①^{-1} + 0 + ①^{-1} - \frac{(0 + ①^{-1}) \cdot (0 + ①^{-1}) \cdot (0 + ①^{-1})}{6} =$$

$$2①^{-1} - \frac{①^{-3}}{6} = 2①^{-1}(\text{-0.166667})①^{-3}.$$

By applying (10) we have the required values

$$f(0) = 0, \quad f'(0) = 2, \quad f''(0) = 2! \cdot 0 = 0, \quad f^{(3)}(0) = 3! \cdot (\text{-0.166667}) = -1.$$

That, obviously, coincide with the respective analytical derivatives (that, we emphasize this fact again, were not used by the Infinity Computer)

$$f'(x) = 2 - 0.5x^2, \quad f''(x) = -x, \quad f^{(3)}(x) = -1$$

calculated at the point $y = 0$. □

Example 3 Suppose that we have a computer procedure $f(x) = \frac{x \cdot x + 1}{x}$ implementing the function $g(x) = \frac{x^2 + 1}{x}$ and we want to calculate the values $f(y), f'(y), f''(y)$, and $f^{(3)}(y)$ at a point $y = 3$. We consider the Infinity Computer that returns grossdigits corresponding to the exponents of grossone from 0 to -3. Then we have

$$f(3 + ①^{-1}) = \frac{(3 + ①^{-1}) \cdot (3 + ①^{-1}) + 1}{3 + ①^{-1}} = \frac{10①^06①^{-1}1①^{-2}}{3①^01①^{-1}} =$$

$$3.333333①^00.888889①^{-1}0.037037①^{-2} - 0.0123457①^{-3}.$$

By applying (10) we obtain that

$$f(3) = 3.333333, \quad f'(3) = 0.888889,$$

$$f''(3) = 2! \cdot 0.037037 = 0.074074, \quad f^{(3)}(3) = 3! \cdot (-0.0123457) = -0.074074,$$

that are values which one obtains by using explicit analytic formulae

$$f(x) = \frac{x^2 + 1}{x}, \quad f'(x) = 1 - x^{-2}, \quad f''(x) = 2x^{-3}, \quad f^{(3)}(x) = -6x^{-4}$$

for $f(x)$ and its derivatives at the point $x = 3$. □

References

1. M. Berz. Automatic differentiation as nonarchimedean analysis. In *Computer Arithmetic and Enclosure Methods*, pages 439–450. Elsevier, Amsterdam, 1992.
2. C. Bischof and M. Bücker. Computing derivatives of computer programs. In *Modern Methods and Algorithms of Quantum Chemistry Proceedings*, NIC Series, vol. 3, pages 315–327. John von Neumann Institute for Computing, Jülich, 2 edition, 2000.
3. L.O. Chua, C.A. Desoer, and E.S. Kuh. *Linear and Non linear Circuits*. MacGraw Hill, Singapore, 1987.
4. J.S. Cohen. *Computer Algebra and Symbolic Computation: Mathematical Methods*. A K Peters, Ltd., Wellesley, MA, 1966.
5. G. Corliss, C. Faure, A. Griewank, L. Hascoet, and U. Naumann, editors. *Automatic Differentiation of Algorithms: From Simulation to Optimization*. Springer-Verlag, New York, 2002.
6. P. Daponte, D. Grimaldi, A. Molinaro, and Ya.D. Sergeyev. An algorithm for finding the zero-crossing of time signals with lipschitzian derivatives. *Measurement*, 16:37–49, 1995.
7. H.Y-F. Lam. *Analog and Digital Filters-Design and Realization*. Prentice Hall Inc., New Jersey, 1979.
8. J.N. Lyness and C.B. Moler. Numerical differentiation of analytic functions. *SIAM J. Numer. Anal.*, 4:202–210, 1967.
9. P. Moin. *Fundamentals of Engineering Numerical Analysis*. Cambridge University Press, Cambridge, 2001.
10. J.M. Muller. *Elementary functions: algorithms and implementation*. Birkhäuser, Boston, 2006.
11. P.M. Pardalos and M.G.C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, New York, 2002.
12. Ya.D. Sergeyev. *Arithmetic of Infinity*. Edizioni Orizzonti Meridionali, CS, 2003.
13. Ya.D. Sergeyev. *http://www.theinfinitycomputer.com*. 2004.
14. Ya.D. Sergeyev. A new applied approach for executing computations with infinite and infinitesimal quantities. *Informatica*, 19(4):567–596, 2008.
15. Ya.D. Sergeyev. *Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them*. EU patent 1728149, 2009.
16. Ya.D. Sergeyev. Numerical computations and mathematical modelling with infinite and infinitesimal numbers. *Journal of Applied Mathematics and Computing*, 29:177–195, 2009.
17. Ya.D. Sergeyev. Numerical point of view on Calculus for functions assuming finite, infinite, and infinitesimal values over finite, infinite, and infinitesimal domains. *Nonlinear Analysis Series A: Theory, Methods & Applications*, 71(12):e1688–e1707, 2009.
18. Ya.D. Sergeyev, P. Daponte, D. Grimaldi, and A. Molinaro. Two methods for solving optimization problems arising in electronic measurements and electrical engineering. *SIAM J. Optim.*, 10(1):1–21, 1999.
19. Ya.D. Sergeyev and D.E. Kvasov. *Diagonal global optimization methods*. Fizmatlit, Moscow, 2008. in Russian.
20. R.G. Strongin and Ya.D. Sergeyev. *Global Optimization and Non-Convex Constraints: Sequential and Parallel Algorithms*. Kluwer Academic Publishers, Dordrecht, 2000.
21. D.F. Walnut. *An Introduction to Wavelet Analysis*. Birkhäuser, Boston, 2004.
22. H.S. Wilf. *Generatingfunctionology*. A K Peters, Ltd., Wellesley, MA, 3rd edition, 2006.
23. M.A. Wolfe. On first zero crossing points. *Applied Mathematics and Computation*, 150:467–479, 2004.