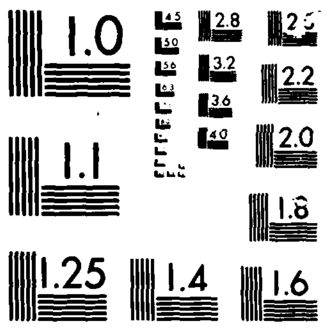
AD-A167 494 OPTIMAL IMPLEMENTATION OF DIFFERENTIATION ARITHMETIC (U) WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER L B RALL MAR 86 MRC-TSR-2920 DAAG29-80-C-0041 1/1

UNCLASSIFIED F/G 12/1 NL



MICROCOP CHART

(2)

AD-A167 494

MRC Technical Summary Report #2920

OPTIMAL IMPLEMENTATION OF DIFFERENTIATION ARITHMETIC

L. B. Rall

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53705

March 1986

(Received February 5, 1986)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

OPTIMAL IMPLEMENTATION OF DIFFERENTIATION ARITHMETIC

L. B. Rall

Technical Summary Report #2920
March 1986

ABSTRACT

Differentiation arithmetic is an ordered-pair arithmetic which evaluates both the value and derivative of functions defined by formulas or subroutines, without symbolics or approximations. As in the case of complex arithmetic, multiplication and division are defined in terms of several real operations. Algorithms are given for evaluation of these operations with the same accuracy as real multiplication and division, that is, to the closest floating-point number. The same kind of optimal implementation is described for Taylor arithmetic, which permits calculation of Taylor coefficients of arbitrary order for functions defined by formulas or subroutines.

AMS (MOS) Subject Classifications: 13F25, 65D25, 65G05, 68Q40

Key words: Automatic differentiation, differentiation arithmetic, optimal computer arithmetic.

Work Unit Number 3 (Numerical Analysis and Scientific Computing)

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

It is commonly believed that evaluation of derivatives of a function requires symbolic manipulation or numerical approximation. However, for functions defined by formulas or computer subroutines, evaluation of the function using an ordered-pair arithmetic called differentiation arithmetic yields exact values of both the function and its derivative at a given value of the variable. In order to implement differentiation arithmetic with maximum accuracy on a computer, special algorithms have to be used for the derivative components of multiplication and division, since these are defined by several real operations, and hence are subject to more roundoff error than the corresponding real operations. Algorithms are given which implement multiplication and division in differentiation arithmetic to the same accuracy as real arithmetic, each component of the result suffering a single roundoff error. This allows calculation of derivatives with the same level of roundoff error as function values. For higher derivatives (more precisely, Taylor coefficients), an arithmetic of $(n+1)$-tuples called Taylor arithmetic can be used in the same way as differentiation arithmetic for the first derivative. The optimal implementation of Taylor arithmetic is indicated, based on well-known algorithms in the theory of optimal computer arithmetic.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

OPTIMAL IMPLEMENTATION OF DIFFERENTIATION ARITHMETIC

L. B. Rall

1. Automatic Differentiation

The process of *automatic differentiation* [8] consists essentially of the evaluation of functions defined by formulas or finite algorithms using rules for operations and representations of variables and constants pertaining to *differentiation arithmetic* [11] rather than ordinary real arithmetic. This process is similar to the evaluation of a function $f : \mathbf{R} \to \mathbf{R}$ given by a formula, such as

$$f(x) = \frac{(x-1)\cdot(x+3)}{x+2} \tag{1.1}$$

in complex instead of real arithmetic to obtain the complex value $f(z)$ of the function $f : \mathbf{C} \to \mathbf{C}$ defined by (1.1) with the real variable x replaced by the complex variable z. This reinterpretation of (1.1) as a complex function can be expressed by saying that its symbols have been "overloaded" by their corresponding meanings as complex operators, variables and constants.

Evaluation of (1.1) in differentiation arithmetic is carried out by a similar overloading. Like complex arithmetic, differentiation arithmetic is an *ordered-pair* arithmetic, with elements of the form $U = (u, u')$, where $u, u' \in \mathbf{R}$. Complex numbers, of course, are also usually represented on a computer by pairs $z = (x, y)$, where $x, y \in \mathbf{R}$ and $z = x + iy$. The *rules* of differentiation arithmetic are as follows:

$$U + V = (u, u') + (v, v') = (u + v, u' + v'), \tag{1.2}$$

$$U - V = (u, u') - (v, v') = (u - v, u' - v'), \tag{1.3}$$

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

(1.4) $$U \cdot V = (u,u') \cdot (v,v') = (u \cdot v, u \cdot v' + v \cdot u'),$$

(1.5) $$U/V = (u,u')/(v,v') = (u/v, (u' - u \cdot v'/v)/v), \quad v \neq 0.$$

The operations *inside* the parentheses in the above definitions are, of course, real operations on real numbers. It is easy to recognize that the first components of the results follow the rules of real arithmetic, while the second components implement the rules for differentiation of the results of the corresponding operations, given that u, v represent values of functions, and u', v' values of their derivatives at some real x. Since

(1.6) $$\frac{dx}{dx} = 1, \qquad \frac{dc}{dx} = 0,$$

where x denotes the independent variable and c a constant, it follows that if x in (1.1) is replaced by $X = (x, 1)$, and the constants 1,2,3 by (1,0), (2,0) and (3,0) respectively, then evaluation using the rules of differentiation arithmetic gives

(1.7) $$f(X) = f((x,1)) = (f(x), f'(x)),$$

where $f(x)$ and $f'(x)$ denote respectively the value of the real function $f : \mathbf{R} \to \mathbf{R}$ and its derivative $f' : \mathbf{R} \to \mathbf{R}$ at the real value x.

Denoting the set $\mathbf{R}^2$ with the rules of arithmetic (1.2)-(1.5) by $\mathbf{D}$, the result (1.6) can be viewed as overloading the symbols of (1.1) to obtain the corresponding mapping $f : \mathbf{D} \to \mathbf{D}$ in the same way the complex extension of f was obtained. From an algebraic standpoint, $\mathbf{D}$ is a division ring with an identity element [11].

This process is called automatic differentiation to distinguish it from symbolic differentiation, which results in a *formula* for $f'(x)$ (which would then have to be evaluated), and numerical differentiation, which produces only an approximate value for $f'(x)$. Automatic differentiation requires only the formula or algorithm for f, and obtains exact values for $f(x)$ and $f'(x)$. This combines the advantages of symbolic and numerical differentiation, while avoiding their disadvantages (an extra evaluation process for $f'(x)$ in the symbolic case, or approximation error if the differentiation is done numerically).

It is interesting to note that the rules for addition and subtraction in $\mathbf{D}$ are the same as for the complex numbers $\mathbf{C}$, while the rules (1.4) and (1.5) for multiplication and division in $\mathbf{D}$ are respectively simpler than the corresponding rules in $\mathbf{C}$. The identity element of addition in $\mathbf{D}$ is $\mathbf{0} = (0,0)$, while the identity for multiplication is $\mathbf{1} = (1,0)$. Elements of $\mathbf{D}$ of the form $(0,u')$ are called *nonunits*. The nonunits are divisors of zero, because $(0,u') \cdot (0,v') = (0,0)$ for arbitrary u', v'. However, if $(u,u') \cdot (v,v') = (0,0)$ and $u \neq 0$, then $(v,v') = (0,0)$ [11].

Automatic differentiation is not limited to rational functions. In the notation introduced above, the chain rule of calculus can be expressed as

$$f(U) = f((u,u')) = \big(f(u), u' \cdot f'(u)\big), \tag{1.8}$$

where $f : \mathbf{R} \to \mathbf{R}$ is a differentiable real function and $U \in \mathbf{D}$. Thus, for example, the exponential function is defined by

$$e^{U} = e^{(u,u')} = \big(e^{u}, u' \cdot e^{u}\big),$$

and so on [8], [11]. However, the object of this paper is to focus on optimal computer implementation of the four arithmetic operations (1.2)–(1.5) in $\mathbf{D}$.

2. Computer Arithmetic

The discussion here follows the general theory of computer arithmetic as given in [6]. On a computer, one works with a finite subset $\mathbf{S} \subset \mathbf{R}$, which is called the set of floating-point numbers. The elements of $\mathbf{S}$ may vary from one machine to another, but $\mathbf{S}$ is assumed to satisfy certain natural conditions in each case, namely that $\mathbf{S}$ contains 0,1, and the negative $-s$ of each $s \in \mathbf{S}$ [6].

In addition to the real numbers, most mathematical systems which appear in numerical mathematics can be constructed as Cartesian products of a suitable number of copies of $\mathbf{R}$, with appropriate definitions of operations in terms of real arithmetic. For example, the complex numbers $\mathbf{C}$ and elements of $\mathbf{D}$ can be considered to be ordered pairs of real numbers, that is, elements of $\mathbf{R} \times \mathbf{R}$. The usual representation of these systems on a computer is by corresponding Cartesian products of $\mathbf{S}$. Thus, ordered pairs of floating-point numbers (elements of $\mathbf{S} \times \mathbf{S}$) are the elements of the computer representations $\mathbf{CS}$ and $\mathbf{DS}$ of $\mathbf{C}$ and $\mathbf{D}$, respectively. Representations $\mathbf{MS} \subset \mathbf{M}$ of other numerical mathematical systems $\mathbf{M}$ in terms of floating-point numbers can be constructed in a similar way. It is assumed that $\mathbf{MS}$ contains the negatives of each of its elements, and the identity elements for addition and multiplication of elements of $\mathbf{M}$.

In mathematical systems $\mathbf{M}$ which are constructed by taking Cartesian products of $\mathbf{R}$, each element $m \in \mathbf{M}$ consists of *components* m_i, each of which is a real number. Thus, it is possible to introduce a partial ordering $\leq$ of $\mathbf{M}$ componentwise, that is, for $m, n \in \mathbf{M}$,

$$(2.1) \qquad m \leq n \quad \Longleftrightarrow \quad m_i \leq n_i \quad \forall i.$$

The same definition provides a partial ordering of the floating-point representation $\mathbf{MS}$ of $\mathbf{M}$ because $\mathbf{MS} \subset \mathbf{M}$. This allows the definition of *rounding* from $\mathbf{M}$ to $\mathbf{MS}$, which is a mapping $\square: \mathbf{M} \to \mathbf{MS}$ with the properties of a semimorphism [6]:

$$(2.2) \quad \begin{cases} \square m = m, & \forall m \in \mathbf{MS}, \\ m \leq n \Longrightarrow \square m \leq \square n, & \forall \text{ comparable } m, n \in \mathbf{M}, \\ \square(-m) = -\square m, & \forall m \in \mathbf{M}. \end{cases}$$

In the case of a rounding from $\mathbf{R}$ to $\mathbf{S}$ satisfying (2.2), the second condition implies that there are no floating point numbers (elements of $\mathbf{S}$) between r and $\square r$ for each $r \in \mathbf{R}$. In this sense, the rounding is said to be of *maximal* accuracy. The case most commonly considered of rounding from $\mathbf{R}$ to $\mathbf{S}$ is the rounding of an element $r \in \mathbf{R}$ to the closest element of $\mathbf{S}$ to r, with ties broken by a rule which satisfies the third condition of (2.2). This rounding, which will be denoted by $\bigcirc$, is said to be of *maximum* accuracy. The ideas of maximal and maximum accuracy of roundings apply componentwise to mathematical systems $\mathbf{M}$ which are products of $\mathbf{R}$. Thus, there will be no elements of $\mathbf{S}$ between m_i and $\square m_i$ in the first case, and $\bigcirc m_i$ is the closest element of $\mathbf{S}$ to each component m_i of $m \in \mathbf{M}$ in the second.

The concept (2.2) of a semimorphism provides a rational way in which to define operations in $\mathbf{MS}$ on the basis of the corresponding operations in $\mathbf{M}$. Suppose, for example, that $\star$ denotes a binary operator in $\mathbf{M}$, for example, one of the arithmetic operators $+, -, \cdot, /$. Then, the corresponding operator in $\mathbf{MS}$, denoted by $\boxed{\star}$, is defined by

$$(2.3) \qquad m \boxed{\star} n = \square(m \star n), \qquad \forall m, n \in \mathbf{MS}.$$

Implementation of the operator $\star$ in $\mathbf{MS}$ in this way is said to be *optimal* with respect to the rounding $\square$.

3. Implementation of Differentiation Arithmetic

A method for the optimal implementation of differentiation arithmetic with respect to the rounding $\bigcirc : \mathbf{D} \to \mathbf{DS}$ of maximum accuracy will now be described. First of all, suppose that the rounding $\bigcirc : \mathbf{R} \to \mathbf{S}$ and corresponding optimal arithmetic operators are available. Then, operators $+, -, \cdot, /$ can be defined for $U, V \in \mathbf{DS}$ by

$$(3.1) \qquad \begin{cases} U + V := (u \oplus v,\ u' \oplus v'), \\ U - V := (u \ominus v,\ u' \ominus v'), \\ U \cdot V := (u \odot v,\ u \odot v' \oplus v \odot u'), \\ U/V := \big(u \oslash v,\ (u' \ominus u \odot v' \oslash v) \oslash v\big), \qquad v \neq 0. \end{cases}$$

The above is an example of the "vertical" definition of computer arithmetic [6] in a mathematical system in which arithmetic is defined componentwise in terms of real arithmetic. Even though the rounding $\bigcirc : \mathbf{R} \to \mathbf{S}$ of maximum accuracy is used, one has in general that

$$(3.2) \qquad \begin{aligned} \bigcirc(u \cdot v' + v \cdot u') &\neq u \odot v' \oplus v \odot u', \\ \bigcirc\big((u' - u \cdot v'/v)/v\big) &\neq (u' \ominus u \cdot v' \oslash v) \oslash v. \end{aligned}$$

Hence, the computer implementation (3.1) of differentiation arithmetic, while easy to program for applications [8], [9], [10], is suboptimal. Special algorithms are required for the implementation of multiplication and division in **DS**.

Optimal implementation of multiplication can be accomplished by use of the optimal scalar product of real vectors, which is required by the general theory of computer arithmetic. That is, if $x, y \in \mathbf{S}^n$, say $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, then their scalar product is the real number

$$(3.3) \qquad x * y = \sum_{i=1}^{n} x_i \cdot y_i,$$

sometimes also called the dot product, or inner product, of x and y. Optimal implementation of this product with respect to the rounding $\bigcirc$ requires the operator $\circledast$ defined by

$$(3.4) \qquad x \circledast y := \bigcirc\Big(\sum_{i=1}^{n} x_i \cdot y_i\Big), \qquad \forall x, y \in \mathbf{S}^n.$$

According to the definition of . . this means that the closest element of **S** to the sum of products on the right sides of (3.3) and (3.4) has to be calculated. It turns out that implementation of $\circledast$ is not difficult [1], and this operator is already available in Pascal-SC and the IBM product ACRITH. Thus, an optimal implementation of multiplication in **DS** with respect to $\bigcirc$ is given by

$$(3.5) \qquad U \cdot V := \big(u \odot v,\ (u,v) \circledast (v',u')\big), \qquad \forall\, U, V \in \mathbf{DS}.$$

Division in **DS** requires a more careful approach. Setting $W = (w, w') = U/V$, one can write

$$(3.6) \qquad w = u/v, \qquad w' = \frac{v \cdot u' - u \cdot v'}{v^2},$$

as in elementary calculus. The optimal rounding $\bigcirc w = u \oslash v$ of the first component w of W presents no problem. The numerator of w' can be computed to the closest floating-point number by using the optimal scalar product $(u,v) \circledast (-v', u')$, as in the case of multiplication, and $\bigcirc(v^2) = v \odot v$. Hence, the numerator and denominator of w' can each be computed to the closest floating-point number, but their quotient can differ slightly from $\bigcirc w'$ [2]. There is also a practical problem, which also arises in the algorithm for optimal complex division [13], namely, that overflow or underflow can occur in the computation of the numerator or denominator, even though the exact result w' can be represented with maximum accuracy by an element of **S**. This problem can be handled by the introduction of appropriate scale factors, followed by calculation of the scaled result to maximum accuracy, and then examination of the scale factors to determine if overflow or underflow will actually occur [13]. If not, then a good approximation w'_r to $\bigcirc w'$ is obtained in this way.

In order to correct w_0', if necessary, the method of iterative refinement of the solution of a linear systems of equations is used [2], [3], [12]. Note that $W = U/V$ is equivalent to $V \cdot W = U$, which gives the lower triangular system of equations

$$\begin{aligned} v \cdot w &= u, \\ v' \cdot w + v \cdot w' &= u', \end{aligned} \tag{3.7}$$

for the components w, w' of W in terms of the components of U, V. Thus, given approximations w_0, w_0' to the solutions of (3.7), for $w = w_0 + \epsilon$ and $w' = w_0' + \epsilon'$, one has

$$\begin{aligned} \epsilon &= (u - v \cdot w_0)/v, \\ \epsilon' &= (u' - v \cdot w_0' - v' \cdot w_0 - v' \cdot \epsilon)/v. \end{aligned} \tag{3.8}$$

Since w_0 has been computed to maximum accuracy, ϵ is negligible with respect to w_0, but could effect the value of ϵ'. Again, the numerator of ϵ' is calculated by use of the optimal scalar product, giving

$$w_1' := w_0' \oplus \left((u', -v, -v') \circledast (1, w_0', \epsilon)\right) \oslash v. \tag{3.9}$$

In this case, one correction is sufficient [2], and one has

$$\bigcirc w = w_0, \qquad \bigcirc w' = w_1'. \tag{3.10}$$

In summary, an algorithm for optimal division in **DS** is as follows:

1°. **If** $u = 0$ **then**

$$w_0 := 0, \qquad w_1' := u' \oslash v, \tag{3.11}$$

else compute

$$w_0 := u \oslash v, \qquad w_0' := \left((u, v) \circledast (-v', u')\right) \oslash (v \odot v), \tag{3.12}$$

scaling the numerator and denominator of w_0' if necessary to avoid over or underflow; if $w_0' \in$ **S**, **then** calculate w_1' by (3.9);

2°. Set

$$U/V := (w_0, w'_1) \in \mathbf{DS}. \tag{3.13}$$

Thus, it is possible to implement the operators $\star$ in **DS** with maximum accuracy using the corresponding operators $\boxed{\star}$ in **S** and the optimal scalar product $\circledast$. This is an example of the "horizontal" definition of computer arithmetic in the floating-point system **DS** corresponding to the mathematical system **D** of differentiation arithmetic. The suboptimal definition (3.1) of computer arithmetic in **DS** will result in derivatives being computed with a greater number of rounding errors than function values, while the definitions of addition and subtraction in (3.1), multiplication by (3.5), and division by (3.13) will result in equal numbers of rounding errors for both function and derivative evaluations.

The techniques of Böhm [2], [3] for evaluation of arithmetic expressions with maximum accuracy can also be extended from **S** to **DS**, because derivatives of arithmetic operations are again defined by arithmetic expressions in (1.2)–(1.5).

4. Taylor Arithmetic

The definition of differentiation arithmetic given in §1 can be generalized immediately to give values of higher derivatives, or, more conveniently, Taylor coefficients

$$u_k = u^{(k)}(x) \cdot \frac{h^k}{k!}, \qquad k = 0, 1, \ldots, \tag{4.1}$$

of the function $u(x+h)$, adopting the usual convention for $k = 0$ that $u_0 = u(x)$ [8]. The corresponding Taylor polynomial $u_n(x+h)$ of degree n in h is uniquely defined by the vector

$$U = (u_0, u_1, \ldots, u_n) \tag{4.2}$$

in $\mathbf{R}^{n+1}$. As in the case of $\mathbf{D}$, arithmetic operations are defined for such vectors to obtain a mathematical system $\mathbf{T}_n$. Addition and subtraction are the corresponding vector operations:

$$U \pm V = (u_0 \pm v_0, u_1 \pm v_1, \ldots, u_n \pm v_n). \tag{4.3}$$

Multiplication and division are derived from the corresponding operations between Taylor polynomials, with the result truncated to degree n. For multiplication,

$$W = U \cdot V, \qquad w_k = \sum_{i=0}^{k} u_i \cdot v_{k-i}, \qquad k = 0, 1, \ldots, n. \tag{4.4}$$

In the case of division, $W = U/V$ is equivalent to $V \cdot W = U$, so (4.4) can be used to obtain

$$W = U/V, \qquad w_k = \left\{ u_k - \sum_{i=0}^{k-1} w_i \cdot v_{k-i} \right\} \Big/ v_0, \qquad k = 0, 1, \ldots, n, \qquad v_0 \neq 0. \tag{4.5}$$

One sees immediately that $\mathbf{R} = \mathbf{T}_0$, $\mathbf{D} = \mathbf{T}_1$. For $1 \leq n < +\infty$, the $\mathbf{T}_n$ are division rings with identity, the same as $\mathbf{D}$. For $n = \infty$, the arithmetic defined by (4.3)-(4.5) is simply the arithmetic of formal power series (see [5], Chapter 1), and $\mathbf{T}_\infty$ is an integral domain. It follows immediately from the definition of the arithmetic operations in $\mathbf{T}_n$

that evaluation of a rational function $u : \mathbf{R} \to \mathbf{R}$ at $X = (x, h, 0, \ldots, 0) \in \mathbf{T}_n$ gives $U = u(X) = (u_0, u_1, \ldots, u_n) \in \mathbf{T}_n$, where the u_k are the Taylor coefficients (4.1) [8].

Automatic generation of Taylor series is not limited to rational functions. As in the case of $\mathbf{D}$, it is also possible to define standard functions in $\mathbf{T}_n$ by use of well-known recurrence relations for their Taylor series expansions [8]. For example, for $W = e^U = \exp\{U\} = (w_0, w_1, \ldots, w_n)$, one has

$$(4.6) \qquad w_0 = e^{u_0}, \qquad w_k = \sum_{i=0}^{k-1} \left(\frac{k-i}{k}\right) \cdot w_i \cdot u_{k-i}, \qquad k = 1, 2, \ldots, n.$$

The floating-point system corresponding to $\mathbf{T}_n$ will be denoted, as before, by $\mathbf{T}_n\mathbf{S}$. Optimal implementations of $+, -, \cdot$ with respect to the rounding $\bigcirc$ are easily performed using the corresponding operators $\circledast$ in $\mathbf{S}$ and the optimal scalar product $\circledast$. For example, defining

$$(4.7) \qquad U_k = (u_0, u_1, \ldots, u_k), \qquad V_{-k} = (v_k, v_{k-1}, \ldots, v_0),$$

one has

$$(4.8) \qquad U \cdot V := \left(u_0 \odot v_0, U_1 \circledast V_{-1}, U_2 \circledast V_{-2}, \ldots, U_n \circledast V_{-n}\right).$$

Just as in $\mathbf{DS}$, division is the only arithmetic operator which requires a special algorithm for its optimal implementation in $\mathbf{T}_n\mathbf{S}$. A good, but suboptimal, implementation can be made immediately on the basis of (4.5) [4]. Define the vectors $V'_{-k} \in \mathbf{S}^k$ by

$$(4.9) \qquad V'_{-k} = (v_k, v_{k-1}, \ldots, v_1).$$

Then, (4.5) can be approximated closely by

$$(4.10) \qquad w_0 := u_0 \oslash v_0, \qquad w_k := \left(u_k - W_{k-1} \circledast V'_k\right) \oslash v_0, \qquad k = 1, 2, \ldots, n.$$

An algorithm for optimal division in $\mathbf{T}_n\mathbf{S}$ can be developed as in $\mathbf{DS}$ by noting that $W = U/V$ is equivalent to $V \cdot W = U$, which in turn is equivalent to the lower triangular system of equations

$$
(4.11) \qquad
\begin{aligned}
v_0 \cdot w_0 &= u_0, \\
v_1 \cdot w_0 + v_0 \cdot w_1 &= u_1, \\
&\vdots \\
v_n \cdot w_0 + v_{n-1} \cdot w_1 + \ldots + v_0 \cdot w_n &= u_n.
\end{aligned}
$$

Using the values (4.10) as initial approximations, the method of iterative residual correction can then applied to the system (4.11) to obtain values of $w_0, w_1, \ldots, w_n \in \mathbf{S}$ of maximal accuracy [2], [3], [12]. Thus, the calculation of Taylor coefficients of a function can be performed at the same level of roundoff error as the calculation of the value of the function. Furthermore, since the Taylor coefficients of the results of arithmetic operations are rational functions of the operand values, it is also possible to extend the method given by Bōhm [2], [3] for the evaluation of rational functions to maximal accuracy to the evaluation of Taylor coefficients of rational functions to maximal accuracy.

References

[1] G. Bohlender and K. Grüner. Realization of an optimal computer arithmetic, [7], pp. 247-268 (1983).

[2] H. Böhm. Berechnung von Polynomnullstellen und Auswertung arithmetischer Ausdrücke mit garantierter maximaler Genauigkeit. Dissertation, University of Karlsruhe, 1983.

[3] H. Böhm. Evaluation of algebraic expressions with maximum accuracy, [7], pp. 121-137 (1983).

[4] G. Corliss and L. B. Rall. Automatic generation of Taylor coefficients in Pascal-SC: Basic operations and applications to ordinary differential equations, *Transactions of the First Army Conference on Applied Mathematics and Computing*, pp. 177-209. U. S. Army Research Office, Research Triangle Park, N.C., 1984.

[5] P. Henrici. Applied and Computational Complex Analysis, Vol. 1. Wiley, New York, 1974.

[6] U. W. Kulisch and W. L. Miranker. Computer Arithmetic in Theory and Practice. Academic Press, New York, 1981.

[7] U. W. Kulisch and W. L. Miranker (Eds.). A New Approach to Scientific Computation. Academic Press, New York, 1983.

[8] L. B. Rall. Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science No. 120. Springer-Verlag, Berlin-Heidelberg-New York, 1981.

[9] L. B. Rall. Differentiation and generation of Taylor coefficients in Pascal-SC, [7], pp. 291-309 (1983).

[10] L. B. Rall. Differentiation in Pascal-SC: Type GRADIENT. *ACM Trans. Math. Software* **10**, no. 2 (1984), 161-184.

[11] L. B. Rall. The arithmetic of differentiation. MRC Technical Summary Report No. 2688, Mathematics Research Center, University of Wisconsin-Madison, 1984. To ap-

pear in *Mathematics Magazine*, February, 1986.

[12] S. Rump. Solving algebraic problems with high accuracy, [7], pp. 51–120 (1983).

[13] J. Wolff von Gudenberg. Einbettung algemeiner Rechnerarithmetik in Pascal mittels eines Operatorkonzeptes und Implementierung der Standardfunktionen mit optimaler Genauigkeit. Dissertation, University of Karlsruhe, 1980.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 2920	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) OPTIMAL IMPLEMENTATION OF DIFFERENTIATION ARITHMETIC		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) L. B. Rall		8. CONTRACT OR GRANT NUMBER(s) DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, Wisconsin 53705		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit Number 3 - Numerical Analysis and Scientific Computing
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709		12. REPORT DATE March 1986
		13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Automatic differentiation
Differentiation arithmetic
Optimal computer arithmetic

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Differentiation arithmetic is an ordered-pair arithmetic which evaluates both the value and derivative of functions defined by formulas or subroutines, without symbolics or approximations. As in the case of complex arithmetic, multiplication and division are defined in terms of several real operations. Algorithms are given for evaluation of these operations with the same accuracy as real multiplication and division, that is, to the closest floating-point number. The same kind of optimal implementation is described for Taylor arithmetic, which permits calculation of Taylor coefficients of arbitrary order for functions defined by formulas or subroutines.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END
FILMED

6-86

DTIC