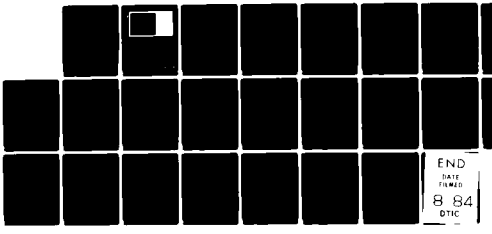
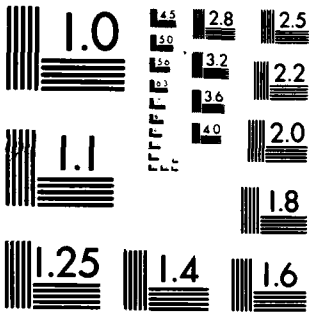
AD-A142 964 THE ARITHMETIC OF DIFFERENTIATION(U) WISCONSIN 1/1
UNIV-MADISON MATHEMATICS RESEARCH CENTER L B RALL
MAY 84 MRC-TSR-2688 DAAG29-80-C-0041

UNCLASSIFIED F/G 12/1 NL





MRC Technical Summary Report #2688

THE ARITHMETIC OF DIFFERENTIATION

L. B. Rall

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

May 1984

(Received May 7, 1984)

Sponsored by

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

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

THE ARITHMETIC OF DIFFERENTIATION

L. B. Rall

Technical Summary Report #2688
May 1984

ABSTRACT

This report describes automatic differentiation, which is neither symbolic nor approximate, for single functions of one real variable. The rules of evaluation and differentiation are combined into an ordered-pair arithmetic similar to complex arithmetic, but slightly simpler. Evaluation of the formula for a function in this arithmetic yields both the values of the function and its derivative, without a formula for the derivative of the function, and without numerical approximation, since this arithmetic is based on the well-known rules for differentiation. The properties of this arithmetic are examined, and illustrated by simple examples. Subroutines are given for differentiation arithmetic both on a hand-held programmable calculator, and in the microcomputer language Pascal-SC. An application of this arithmetic to the solution of equations by Newton's method is given, using a Pascal-SC program.

AMS (MOS) Subject Classifications: 26-01, 26-04, 26A06, 65H05, 68B99

Key Words: Automatic differentiation

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

Most people think that evaluation of derivatives on a computer has to be either symbolic or numerical, that is, approximate. By combining the rules of differentiation with the rules for evaluation, however, it is possible to perform differentiation using only the formula (or subroutine) for the function involved. It is not necessary to produce a formula for the derivative of the function to obtain the exact value of its derivative for any given values of its arguments. The way this method of automatic differentiation works is not immediately obvious, because everyone is taught in school that the first step in evaluating the derivative of a function is to derive the formula for the derivative, and the only alternative to this is to approximate the derivative by a difference quotient. However, the definition of the dervative and the rules derived from it actually define values of the derivative, not formulas, so that these rules can be used to define what is called differentiation arithmetic here for single real-valued functions of one real variable. This is an ordered-pair arithmetic, similar to complex arithmetic but actually a little simpler. When the formula for the function is evaluated in this arithmetic, the value of the function and its derivative are obtained automatically, without symbolics and without approximations. Differentiation arithmetic is easy to program for hand-held calculators as well as computers, and some examples are given.

Automatic differentiation was developed first in the general setting of functions of several variables, and for Taylor series expansions of functions of single variables. The special case considered here was presented to the Wisconsin Section of the Mathematical Association of America in the hope that this simple setting would help to explain away some of the current misconceptions about differentiation on the computer. Automatic differentiation is as accurate as symbolic differentiation, which requires a lot of software, but much faster. It is also suitable for parallel computation. Numerical differentiation gives only approximate answers, while automatic differentiation gives correct results at about the same speed or even faster.

THE ARITHMETIC OF DIFFERENTIATION

L. B. Rall

1. A simple calculus problem. Traditional ways of thinking and doing things exert a strong hold in mathematics, as well as in other human endeavors. For example, generations of students were taught to solve linear systems of equations by using determinants and Cramer's rule [2]. It was not until people actually tried solving linear systems using desk calculators and later computers that more efficient methods based on Gaussian elimination came into general use [7], [3]. Even today, it is disconcerting to encounter people who think that the "only way" to solve linear systems is by the Cramer's rule they learned in school.

Calculus is another subject with a long, rich history and strong traditions. As an example of a traditional approach to a simple problem in calculus, suppose you are given a function defined by a formula, say

$$F(X) = \frac{X^2 + 2X - 3}{X + 2} = \frac{(X - 1)\cdot(X + 3)}{X + 2}\,, \tag{1.1}$$

and you want to find the value of the derivative F'(X) for a given value of X. Most people have been taught to solve this problem by first deriving a formula for the derivative,

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

$$F'(X) = \frac{X^2 + 4X + 7}{(X + 2)^2} = 1 + \frac{3}{(X + 2)^2}\,, \tag{1.2}$$

and then using the usual rules of arithmetic to evaluate F'(X). This method, like Cramer's rule, is theoretically correct, and not complicated for this example. However, this is not the **only** way to calculate the values of derivatives. It turns out that it is possible to obtain both the values F(X) and F'(X) simply by evaluating the formula for F(X) using a different kind of arithmetic, called **differentiation arithmetic**. A formula for F'(X) is not required, contrary to common belief. Differentiation arithmetic is an **ordered-pair** arithmetic, similar to complex arithmetic. The rules for this arithmetic will be defined later; first, it is important to analyze the key idea of function evaluation in ordinary arithmetics, and see what this process has in common with differentiation.

2. Evaluation of functions. The usual way to evaluate a rational function, (1.1) for example, is to break it down into a sequence of additions, subtractions, multiplications, and divisions, and then apply the rules for arithmetic to each. Given a simple formula, this is often done instinctively, without much thought. If a hand-held calculator is being used, however, it is useful to think out the sequence of arithmetic operations being used, and perhaps write them down, especially if a complicated formula is being evaluated. Such a sequence is called a **code list** for F(X) [10], [12]. For example, evaluation of (1.1) can be done in the following steps:

```
                T1 := X - 1;
                T2 := X + 3;
(2.1)           T3 := T1*T2;
                T4 := X + 2;
              F(X) := T3/T4;
```

or some other equivalent sequence. The intermediate quantities T1,T2,T3,T4 are each the result of a single arithmetic operation.

Another way to visualize the evaluation of F(X) is by means of a Kantorovich graph

[5], [11], [12], shown in Figure 2.1.

The Kantorovich graph is a directed graph in which information (numerical values in this case) flows along edges from top to bottom, and the indicated arithmetic operations are performed at the nodes. This graph shows which operations must be performed before others can be done, or, from another point of view, the operations which can be performed independently or in parallel. Various code lists equivalent to (2.1) can be read off the graph. In the early days of computing, the tedious process of preparation of code lists to evaluate functions was done by the programmer. Modern computer languages now contain formula translators, which automatically analyze formulas such as (1.1) by a fixed algorithm and produce sequences of instructions similar to (2.1) for the machine to execute.

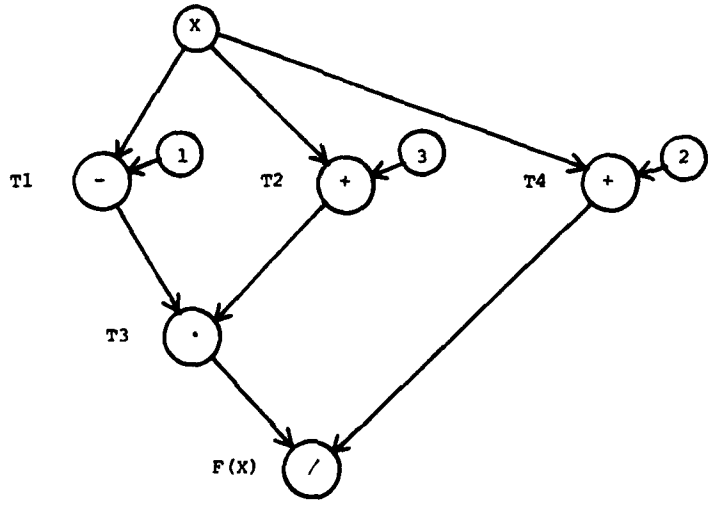


Figure 2.1. A Kantorovich Graph for the Function (1.1).

Once the code list (2.1) has been formed, evaluation of F(X) for some value of X, say X = 3, is a trivial matter; 3 is substituted for X and the indicated sequence of operations are performed:

$$\begin{aligned} T1 &:= 3 - 1 = 2; \\ T2 &:= 3 + 3 = 6; \\ T3 &:= 2 \cdot 6 = 12; \\ T4 &:= 3 + 2 = 5; \\ F(3) &:= 12/5 = 2.4\ . \end{aligned} \tag{2.2}$$

Of course, code lists are not the only way to express evaluation of a function. Another popular method is to express F(X) in "reverse Polish" notation, for example, by

$$F(X) := X\uparrow 1 - X\uparrow 3 + \cdot X\uparrow 2 + /; \tag{2.3}$$

where ↑ represents "entering" the preceding value, or "pushing" it onto a stack.

3. Differentiation of functions. Elementary differential calculus is based on the definition

$$F'(X) = \lim_{\Delta X \to 0} \frac{F(X + \Delta X) - F(X)}{\Delta X}\ . \tag{3.1}$$

However, this definition is seldom used for actual differentiation of a function such as (1.1). Instead, it is used to derive rules for differentiation of sums, differences, products, quotients, and other functions encountered in mathematical analysis. Once these rules have been established, differentiation proceeds in much the same way as function evaluation: The function is expressed as a sequence of additions, subtractions, multiplications, divisions, etc., and the appropriate rules are applied. This technique is valid as a consequence of the chain rule, which is derived from the definition (3.1). To see how this works for our example, apply the rules for differentiation to the code list

(2.1):

(3.2)

$$T1' := 1;$$
$$T2' := 1;$$
$$T3' := T1 \cdot T2' + T2 \cdot T1';$$
$$T4' := 1;$$
$$F'(X) := (T4 \cdot T3' - T3 \cdot T4')/T4^2.$$

This, along with (2.2), gives all the information needed to evaluate F'(3). The steps are:

(3.3)

$$T1' := 1;$$
$$T2' := 1;$$
$$T3' := 2 \cdot 1 + 6 \cdot 1 = 8;$$
$$T4' := 1;$$
$$F'(3) := (5 \cdot 8 - 12 \cdot 1)/5^2 = 28/25 = 1.12 .$$

This gives the correct value of F'(3) without the use of a formula for F'(X). This process is called **automatic differentiation** [11], [12] in order to distinguish it from **symbolic differentiation**, which produces a formula for the derivative. It is also important to note that automatic differentiation is **not** "numerical differentiation", which yields only an approximate value for F'(X), for example, by use of the difference quotient

(3.4)

$$\frac{\Delta F}{\Delta X} = \frac{F(X + \Delta X) - F(X)}{\Delta X} ,$$

for a fixed value of ΔX.

As presented above, the use of automatic differentiation requires the formation of the additional code list (3.2), and evaluation of F'(X) by (3.3) depends on values obtained in the evaluation (2.2) of F(X). The formation of an extra code list is eliminated, and the evaluation of F(X) and F'(X) are combined by the use of differentiation arithmetic applied

to the original code list (2.1).

4. Differentiation arithmetic. The idea behind differentiation arithmetic is t ~ familiar one of an ordered-pair arithmetic. Here, arithmetic operations are defined in the set $S^2 = S \times S$ of pairs of elements of a set S componentwise in terms of already established operations in S. Some examples, with $S = K$, the set of integers, or $S = R$, the set of real numbers, are:

(i) rational arithmetic: $X = p/q, \quad p,q \in K$;

(ii) complex arithmetic: $X = (x,y) = x + iy, \quad x,y \in R$;

(iii) interval arithmetic: $X = [a,b] = \{x \mid a \leq x \leq b, \ a,b,x \in R, \ a \leq b\}$.

The rules of rational and complex arithmetic are well-known; interval arithmetic is described in [1], [8], or [9]. In any of these arithmetics, the function F(X) given by (1.1) can be evaluated by using the code list (2.1), starting with a different type of element X, and using different rules of arithmetic than in the evaluation of F(X) for a real number $X = x \in R$.

The basic elements for differentiation arithmetic are pairs of real numbers

$$F = (f,f'), \ G = (g,g'), \ \ldots \in D = R^2.$$

For the time being, the prime ' is to be regarded only as a marker to distinguish the second element of a pair. The rules of differentiation arithmetic are as follows:

1°. Addition

$$F + G = (f,f') + (g,g') = (f + g, \ f' + g'); \tag{4.1}$$

2°. Subtraction

$$F - G = (f,f') - (g,g') = (f - g, \ f' - g'); \tag{4.2}$$

3°. Multiplication

$$F \cdot G = (f,f') \cdot (g,g') = (f \cdot g, \ f \cdot g' + g \cdot f'); \tag{4.3}$$

4°. Division

$$F / G = (f,f') / (g,g') = \left(f/g, \ \frac{g \cdot f' - f \cdot g'}{g^2}\right), \quad g \neq 0. \tag{4.4}$$

Examination of these rules shows their simple structure: The first components of the

results are formed by the rule for **evaluation** in real arithmetic, and the second by the corresponding rule for **differentiation**, assuming the first components of the operands represent function values, and the second components are values of their derivatives. Two further rules of differentiation are needed in order to represent special quantities in differentiation arithmetic. First, if x denotes a variable, the basic rule is

$$\frac{dx}{dx} = 1, \tag{4.5}$$

from which it follows that the **independent variable** X in differentiation arithmetic is represented by a pair

$$X = (x,1), \tag{4.6}$$

where x is the **value** of the independent variable X. Second, if c denotes a constant, then

$$\frac{dc}{dx} = 0, \tag{4.7}$$

which means that **constants** in differentiation arithmetic are represented by pairs

$$C = (c,0), \tag{4.8}$$

where c is again the **value** of the constant C.

Evaluation of a formula for a rational function F in differentiation arithmetic should give the correct values for both F(X) and F'(X), because this kind of function can be expressed as a sequence of arithmetic operations and the chain rule holds. To see how differentiation arithmetic works, try it out on the example (1.1), following the code list (2.1). Here, it is convenient to identify the literal constants appearing in (2.1) with pairs of the form (4.8). The resulting **mixed** arithmetic (similar to multiplication of vectors by scalars) is less cumbersome than conversion of constants to the form (4.8).

With X = (3,1), (2.1) becomes

$$
\begin{aligned}
&T1 := (3,1) - 1 = (2,1);\\
&T2 := (3,1) + 3 = (6,1);\\
&T3 := (2,1)\cdot(6,1) = (2\cdot 6,\ 2\cdot 1 + 6\cdot 1) = (12,8);\\
&T4 := (3,1) + 2 = (5,1);\\
&F((3,1)) := (12,8)/(5,1) = (12/5,\ (5\cdot 8 - 12\cdot 1)/5^2) = (12/5,\ 28/25);
\end{aligned}
\tag{4.9}
$$

so that

$$F(3) = 12/5 = 2.4, \quad F'(3) = 28/25 = 1.12 \tag{4.10}$$

for the real-valued function F(X) defined by (1.1).

The rules of differentiation arithmetic are actually simpler than the rules for complex arithmetic, and give the correct values for rational functions and their derivatives without symbolic manipulations or resort to numerical approximations. Furthermore, differentiation arithmetic is easy to program for hand-held programmable calculators (see Appendix A) or computers [6], [12], [13] (see Appendix B).

5. Properties of differentiation arithmetic. The reason why diffentiation arithmetic works when applied to rational functions is based on the fact that it has certain key properties in common with real and complex arithmetic. Differentiation arithmetic takes place in the mathematical system **D** consisting of the elements of R^2 with the operations (4.1)-(4.4). It is easy to verify the following properties:

1°. Addition in **D** forms a commutative group. The identity element for addition is

$$\mathbf{0} = (0,0), \tag{5.1}$$

and each element $F \in D$ has the **additive inverse** (or **negative**)

(5.2) $$-F = -(f,f') = (-f, -f').$$

2°. Multiplication in D forms a commutative semigroup with an identity, that is, multiplication is commutative and associative, and

(5.3) $$1 = (1,0)$$

is the identity element for multiplication, $F \cdot 1 = 1 \cdot F = F$ for all $F \in D$.

3°. Multiplication is distributive across addition,

(5.4) $$F \cdot (G + H) = F \cdot G + F \cdot H,$$

for all $F,G,H \in D$.

4°. There are no **divisors of zero**, which means that if

(5.5) $$F \cdot G = 0,$$

and $F \neq 0$, then $G = 0$.

A mathematical system with these properties is called an **integral domain** [4]. Furthermore, every element $F = (f,f') \in D$ such that $f \neq 0$ has the **multiplicative inverse** (or **reciprocal**) $1/F$ given by

(5.6) $$1/F = 1/(f,f') = (1/f, -f'/f^2).$$

It is important that D is an integral domain because this means that the same results will be obtained independently of the order in which equivalent sequences of arithmetic operations are performed. Thus, for example, the example function F(X) can be evaluated by any code list equivalent to (2.1) in real, complex, or differentiation arithmetic, and the results will agree. The real number system **R** and the complex number system **C** do share a

property which D does not have: In R and C, only the additive identity 0 does not have a reciprocal; integral domains in which this holds are called **fields** [4].

6. Functions in D. In addition to rational functions, the student of calculus is introduced to a basic set of transcendental functions. The rules for differentiation of these **elementary** functions are based on the definition (3.1) and its consequences, such as the chain rule, implicit differentiation, and logarithmic differentiation. Once the rule for differentiation of a function is known, it can be used to define the corresponding function in D. For example,

$$e^F = e^{(f,f')} = (e^f,\ f' \cdot e^f), \tag{6.1}$$

$$\ln F = \ln (f,f') = (\ln f,\ f'/f), \tag{6.2}$$

$$\sin F = \sin (f,f') = (\sin f,\ f' \cdot \cos f), \tag{6.3}$$

$$\cos F = \cos (f,f') = (\cos f,\ -f' \cdot \sin f), \tag{6.4}$$

$$\tan^{-1} F = \tan^{-1} (f,f') = (\tan^{-1} f,\ f'/(1 + f^2)), \tag{6.5}$$

and so on. The **power function** is usually handled in three cases:

$$F^c = (f,f')^c = (f^c,\ f' \cdot c \cdot f^{c-1}), \tag{6.6}$$

$$c^F = c^{(f,f')} = (c^f,\ f' \cdot \ln c \cdot c^f), \tag{6.7}$$

$$F^G = (f,f')^{(g,g')} = (f^g,\ f' \cdot g \cdot f^{g-1} + g' \cdot \ln g \cdot f^g), \tag{6.8}$$

where $c \in R$ denotes a constant.

By following the pattern of the above examples, it is easy to define functions in D

which give the values of the corresponding function and its derivative in $\mathbf{R}$. In general, by the chain rule,

$$u(F) = u((f,f)) = (u(f),\ u'(f)\cdot f'), \tag{6.9}$$

where $u: \mathbf{R} \to \mathbf{R}$ is any differentiable function. The rules of automatic differentiation can thus be extended to a wide class of functions, and in particular those differentiable functions encountered in elementary analysis.

In Appendix A, a set of subroutines for the arithmetic operations (4.1)-(4.4) and the elementary functions (6.1)-(6.5) are given for a typical hand-held programmable RPN calculator, the HP-15C. A similar set of operators and the functions (6.1)-(6.5) are given in Appendix B in the computer language Pascal-SC [14], which can be used on many microcomputers.

7. The geometry of differentiation. It is natural to identify the elements $F = (f,f')$ of $\mathbf{D}$ with the points of the real plane $\mathbf{R}^2$. In this geometry, constants lie on the f-axis ($f' = 0$), and the locus of the independent variable is the line $f' = 1$. A function $F: \mathbf{D} \to \mathbf{D}$ can be parameterized by the real variable x by setting $X = (x,1)$ to obtain

$$F(X) = F((x,1)) = (f(x),f'(x)). \tag{7.1}$$

The graph of F in the f,f'-plane (the so-called **phase plane**) reveals some interesting facts about the underlying real function $f: \mathbf{R} \to \mathbf{R}$. Points at which the graph of F crosses the f-axis ($f' = 0$) correspond to **critical points** of the function f, and crossings of the f'-axis ($f = 0$) to **zeros** of f.

The function

$$f(x) = x^2 + 4 \tag{7.2}$$

is represented by

(7.3) $$F(X) = (x^2 + 4,\ 2x)$$

in D, which is equivalent to

(7.4) $$f = \tfrac{1}{4}f'^2 + 4$$

in the phase plane. The point F = (4,0) is critical, and is furthermore a global minimum of f, since the entire graph of F(X) lies to the right of this point, that is, f > 4, see Figure 7.1.

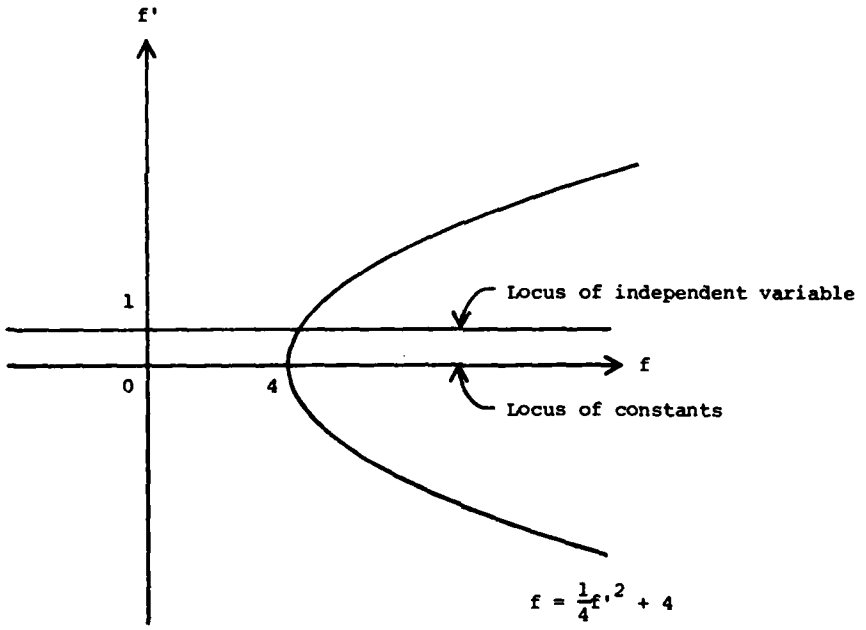


Figure 7.1. Graph of $f(x) = x^2 + 4$ in the Phase Plane.

8. Symbolic differentiation revisited. Symbolic and automatic differentiation have different goals. The purpose of symbolic differentiation is to produce a **formula** for the function $f': \mathbf{R} \to \mathbf{R}$ which is the derivative of the given function $f: \mathbf{R} \to \mathbf{R}$, while automatic differentiation gives the **value** $f'(x)$ of the derivative f' at some given $x \in \mathbf{R}$ (as well as the value $f(x)$). Both use the same techniques, derived from the basic definition (3.1). It is worthwhile to note that (3.1) and the rules derived from it actually define the **value** of the derivative $F'(X)$, and *not* a **formula** for $F'(X)$. The formula is obtained by working with the rules symbolically, and using additional rules for simplification of the symbolic results. With this in mind, the rules of automatic differentiation can also be used for symbolic differentiation. To see this, evaluate the formula (1.1) by using the code list (2.1) with $X = (x,1)$. The results are:

(8.1)

$$
\begin{aligned}
T1 &:= (x,1) - 1 = (x - 1,\ 1);\\
T2 &:= (x,1) + 3 = (x + 3,\ 1);\\
T3 &:= (x-1,1)\cdot(x+3,1) = ((x-1)\cdot(x+3),\ (x-1)\cdot 1 + (x+3)\cdot 1)\\
&\qquad = (x^2 + 2x - 3,\ 2x + 2)\\
T4 &:= (x,1) + 2 = (x + 2,\ 1);\\
F(X) &:= (x^2+2x-3,2x+2)/(x+2,1)\\
&= \left(\frac{x^2 + 2x - 3}{x + 2},\ \frac{(x + 2)\cdot(2x + 2) - (x^2 + 2x - 3)\cdot 1}{(x + 2)^2} \right)\\
&= \left(\frac{x^2 + 2x - 3}{x + 2},\ \frac{x^2 + 4x + 7}{(x + 2)^2} \right).
\end{aligned}
$$

This is the right answer, obtained by applying the rules of automatic differentiation in symbolic form. This example illustrates the inherent difficulty of symbolic differentiation, for example, in the calculation of T3, it has to be recognized that $(x-1)\cdot 1 + (x+3)\cdot 1$ and $2x + 2$ symbolize the same quantity, and the latter form is preferable. In the numerical calculation of T3 for $x = 3$, the arithmetic calculation $2\cdot 1 + 6\cdot 1 = 8$ required no such simplification. Computer programs which do symbolic differentiation are

large and consequently slow compared to the small and fast subroutines required for automatic differentiation. This is because many extra rules for algebraic simplification must be programmed for the symbolic programs, and these are not required for simple numerical evaluation of functions and derivatives.

9. Extensions of differentiation arithmetic. There are several directions in which the differentiation arithmetic presented here for functions of one real variable can be extended. A differentiation arithmetic can be defined for higher derivatives. In the case of second derivatives, the basic elements are triples

$$F = (f, f', f''), \tag{9.1}$$

the independent variable is represented in the form $X = (x,1,0)$, and constants by $C = (c,0,0)$. Leibniz' rule is very useful for the derivation of the rules for multiplication and division [12]. This idea can be extended to obtain derivatives of arbitrary order by evaluation of the formula for the original function, or Taylor coefficients, since rules are known for formation of the (n+1)-tuples

$$F = (f_0, f_1, \ldots, f_n), \tag{9.2}$$

for arithmetic operations and various standard functions, where

$$f_k = \frac{h^k}{k!} f^{(k)}(x), \quad k = 0,1,\ldots,n, \tag{9.3}$$

with the usual conventions $f^{(0)}(x) = f(x)$ and $0! = 1$ [12].

Another extension of differentiation arithmetic is to functions of several variables, that is, to partial derivatives. In this case,

$$F = (f, \nabla f), \tag{9.4}$$

where ∇f is the **gradient vector** of f, that is,

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}\right), \tag{9.5}$$

$x = (x_1, x_2, \ldots, x_n)$. The ith independent variable is represented by $X_i = (x_i, e_i)$, where e_i is the ith unit vector, and constants by $C = (c, 0)$, where 0 denotes the zero vector $0 = (0, 0, \ldots, 0)$. The differentiation arithmetic presented in this paper is actually simply this **gradient** arithmetic for the special case $n = 1$.

Differentiation arithmetic can be extended to higher partial derivatives and Taylor coefficients in several variables in the same way as done for functions of a single variable. Second partial differentiation is based on the representation

$$F = (f, \nabla f, Hf), \tag{9.6}$$

where Hf denotes the **Hessian matrix**

$$Hf(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j}\right) \tag{9.7}$$

of second partial derivatives of f at x.

Increasing the order of derivatives needed or the number of independent variables quickly puts automatic differentiation out of reach of hand calculation. However, the required rules of differentiation arithmetic are easy to program for computers [5], [12], [13], and are well-suited to parallel computation. The results obtained have the accuracy of symbolic differention, but are require much less computation. Furthermore, differentiation arithmetic can be combined with other forms of arithmetic, such as interval arithmetic, to provide information about the values of functions and their derivatives over ranges of values of the variables [8], [9], [12], [13].

10. A simple application of differentiation arithmetic. The use of Newton's method to solve an equation of the form

(10.1) $$f(x) = 0$$

requires values $f(x_n)$ and $f'(x_n)$ of the function and its derivative for each approximation x_n of a solution $x = x^*$ of (10.1). These values are given directly by evaluation of the function f in differentiation arithmetic. For

(10.2) $$X_n = (x_n, 1),$$

the use of differentiation arithmetic gives

(10.3) $$F_n = F(X_n) = (f_n, f_n') = (f(x_n), f'(x_n)),$$

and thus for

(10.4) $$x_{n+1} = x_n - f_n/f_n',$$

(10.5) $$X_{n+1} = (x_{n+1}, 1).$$

Starting with $x_0 = -2.1$, the results for the function (1.1) are:

$$x_0 = 2.10000000000$$
$$F(X_0) = (27.9000000000, 301.000000000),$$
$$x_1 = -2.19269102990$$
$$F(X_1) = (13.3762744874, 81.7975624261),$$
$$x_2 = -2.35622004092$$
$$F(X_2) = (6.06554082423, 24.6420186899),$$
$$x_3 = -2.60236630436$$
$$F(X_3) = (2.37799195653, 9.26798946894),$$

x_4 = -2.85894751483

$F(x_4)$ = (6.33698948666 × 10^{-1}, 5.06619310633)

x_5 = -2.98403136682

$F(x_5)$ = (6.46519385657 × 10^{-2}, 4.09815663218)

x_6 = -2.99980722518

$F(x_6)$ = (7.71210787890 × 10^{-4}, 4.00115698347)

x_7 = -2.99999997213

$F(x_7)$ = (1.11480002330 × 10^{-7}, 4.00000016722)

x_8 = -3.00000000000

$F(x_8)$ = (0.00000000000, 4.00000000000).

These results were computed using the Pascal-SC program given in Appendix C. This program can be modified easily to applied Newton's method to functions other than (1.1).

References

1. G. Alefeld and J. Herzberger. Introduction to Interval Computations. Translated by Jon Rokne. Academic Press, New York, 1983.

2. L. E. Dickson. New First Course in the Theory of Equations. Wiley, New York, 1939.

3. G. E. Forsythe and C. B. Moler. Computer Solution of Linear Algebraic Systems. Prentice-Hall, Englewood Cliffs, N. J., 1967.

4. Peter Henrici. Applied and Computational Complex Analysis, Vol. 1. Wiley, New York, 1974.

5. L. V. Kantorovich. On a mathematical symbolism convenient for performing machine calculations (Russian). Doklady Akad. Nauk SSSR 113 (1957), 738-741.

6. G. Kedem. Automatic differentiation of computer programs. ACM Trans. Math. Software 6 (1980), no. 2, 150-165.

7. C. C. MacDuffee. Theory of Equations. Wiley, New York, 1954.

8. R. E. Moore. Interval Analysis. Prentice-Hall, Englewood Cliffs, N. J., 1966.

9. R. E. Moore. Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics, 2, Philadelphia, 1979.

10. L. B. Rall. Computational Solution of Nonlinear Operator Equations. Wiley, New York, 1969. Reprinted by Krieger, Huntington, N. Y., 1979.

11. L. B. Rall. Applications of software for automatic differentiation in numerical computation. Computing, Suppl. 2 (1980), 141-156.

12. L. B. Rall. Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science No. 120, Springer, New York, 1981.

13. L. B. Rall. Differentiation and generation of Taylor coefficients in Pascal-SC. A New Approach to Scientific Computation, ed. by U. W. Kulisch and W. L. Miranker, pp. 291-309. Academic Press, New York, 1983.

14. L. B. Rall. An introduction to the scientific computing language Pascal-SC. Mathematics Research Center Technical Summary Report #2644. University of Wisconsin-Madison, 1984.

APPENDIX A

Differentiation Arithmetic for a Typical Programmable RPN Calculator

The calculator used for these examples was the HP15-C, manufactured by Hewlett-Packard. The programming is the same or very similar for other machines of this type.

The arithmetic subroutines calculate

$$(h,h') = (f,f')*(g,g'), \text{ where } * \in \{+,-,\cdot,/\}.$$

The sequence of keystrokes to enter F, G is f↑f'↑g↑g', followed by GSB *, where * = A for addition, B for subtraction, C for multiplication, and D for division. The result h will be left in X, and h' in Y. These routines use storage locations 0, 1 and 2.

Addition	Subtraction	Multiplication	Division
f LBL A	f LBL B	f LBL C	f LBL D
X ⇄ Y	X ⇄ Y	STO 0	STO 0
STO 0	STO 0	R↓	R↓
R↓	R↓	STO 1	STO 1
+	-	×	R↓
X ⇄ Y	X ⇄ Y	X ⇄ Y	X ⇄ Y
RCL 0	RCL 0	STO × 1	RCL 1
+	-	RCL 0	÷
g RTN	g RTN	×	STO 2
		+	RCL 0
		RCL 1	×
		g RTN	-
			RCL 1
			÷
			RCL 2
			g RTN

The function subroutines calculate

$$u(f,f') = (u(f),\ u'(f)\cdot f')$$

for the functions u = sqrt, exp, ln, sin, cos, $\tan^{-1}$. The sequence of keystrokes is f↑f', followed by GSB *, where * = 0 (sqrt), 1 (exp), .1 (ln), 2 (sin), 3 (cos), 4 ($\tan^{-1}$). The result u(f) will be left in X, and the derivative u'(f)·f' will be left in Y. These routines use storage location 0.

sqrt(F) = $\sqrt{F}$	exp(F) = e^F	ln(F)
f LBL 0	f LBL 1	f LBL .1
X ↑↓ Y	X ↑↓ Y	X ↑↓ Y
$\sqrt{x}$	g e^x	STO 0
STO 0	STO 0	+
+	×	RCL 0
2	RCL 0	g LN
+	g RTN	g RTN
RCL 0		
g RTN		

sin(F)	cos(F)	$\tan^{-1}$(F)
f LBL 2	f LBL 3	f LBL 4
g RAD	g RAD	g RAD
X ↑↓ Y	X ↑↓ Y	X ↑↓ Y
STO 0	STO 0	STO 0
COS	SIN	g x^2
×	×	1
RCL 0	CHS	+
SIN	RCL 0	+
g RTN	COS	RCL 0
	g RTN	g $\tan^{-1}$
		g RTN

APPENDIX B

Pascal-SC Operators and Functions for Differentiation Arithmetic

Differentiation arithmetic is performed in Pascal-SC by use of the data type DERIV, which is declared as follows:

```
TYPE DERIV = RECORD X,PRIME: REAL END;
```

This means that if F is a variable of type DERIV, then F.X = F(X) represents its function value, and F.PRIME = F'(X) the value of its derivative. The independent variable X has X.PRIME = 1, while C.PRIME = 0 for constants C.

In order to be able to mix REAL, INTEGER, and DERIV types in expressions, where REAL and INTEGER are considered to be constants, 22 arithmetic operators are r required. If K, R, and D denote respectively generic variables of types INTEGER, REAL, and DERIV, then the following operators are needed:

Addition: +D, K + D, D + K, R + D, D + R, D + D;

Subtraction: -D, K - D, D - K, R - D, D - R, D - D;

Multiplication: K*D, D*K, R*D, D*R, D*D;

Division: K/D, D/K, R/D, D/R, D/D.

The following files contain the source code for the above operators:

Addition (DADD.OPR), subtraction (DSUB.OPR), multiplication (DMUL.OPR), d division (DDIV.OPR).

1. ADDITION OPERATORS

```
OPERATOR + (A: DERIV) RES: DERIV;
  BEGIN
    RES := A
  END;

OPERATOR + (K: INTEGER;A: DERIV) RES: DERIV;
  BEGIN
    A.X := K + A.X;
    RES := A
  END;

OPERATOR + (A: DERIV;K: INTEGER) RES: DERIV;
  BEGIN
    A.X := A.X + K;
    RES := A
  END;
```

```
OPERATOR + (R: REAL;A: DERIV) RES: DERIV;
  BEGIN
    A.X := R + A.X;
    RES := A
  END;

OPERATOR + (A: DERIV;R:REAL) RES: DERIV;
  BEGIN
    A.X := A.X + R;
    RES := A
  END;

OPERATOR + (A,B: DERIV) RES: DERIV;
  BEGIN
    A.X := A.X + B.X;
    A.PRIME := A.PRIME + B.PRIME;
    RES := A;
  END;
```

2. SUBTRACTION OPERATORS

```
OPERATOR - (A: DERIV) RES: DERIV;
  BEGIN
    A.X := -A.X;
    A.PRIME := -A.PRIME;
    RES := A
  END;

OPERATOR - (K: INTEGER;A: DERIV) RES: DERIV;
  BEGIN
    A.X := K - A.X;
    A.PRIME := -A.PRIME;
    RES := A
  END;

OPERATOR - (A: DERIV;K: INTEGER) RES: DERIV;
  BEGIN
    A.X := A.X - K;
    RES := A
  END;

OPERATOR - (R: REAL;A: DERIV) RES: DERIV;
  BEGIN
    A.X := R - A.X;
    A.PRIME := -A.PRIME;
    RES := A
  END;

OPERATOR - (A: DERIV;R: REAL) RES: DERIV;
  BEGIN
    A.X := A.X - R;
    RES := A
  END;
```

```
OPERATOR - (A,B: DERIV) RES: DERIV;
  BEGIN
    A.X := A.X - B.X;
    A.PRIME := A.PRIME - B.PRIME;
    RES := A
  END;
```

3. MULTIPLICATION OPERATORS

```
OPERATOR * (K: INTEGER;A: DERIV) RES: DERIV;
  BEGIN
    A.X := K*A.X;
    A.PRIME := K*A.PRIME;
    RES := A;
  END;

OPERATOR * (A: DERIV;K: INTEGER) RES: DERIV;
  BEGIN
    A.X := A.X*K;
    A.PRIME := A.PRIME*K;
    RES := A;
  END;

OPERATOR * (R: REAL;A: DERIV) RES: DERIV;
  BEGIN
    A.X := R*A.X;
    A.PRIME := R*A.PRIME;
    RES := A;
  END;

OPERATOR * (A: DERIV;R: REAL) RES: DERIV;
  BEGIN
    A.X := A.X*R;
    A.PRIME := A.PRIME*R;
    RES := A
  END;

OPERATOR * (A,B: DERIV) RES: DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := A.X*B.X;
    U.PRIME := A.X*B.PRIME + A.PRIME*B.X;
    RES := U
  END;
```

4. DIVISION OPERATORS

```
OPERATOR / (K: INTEGER;A: DERIV) RES: DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := K/A.X;
    U.PRIME := -A.PRIME*U.X/A.X;
    RES := U
  END;
```

```
OPERATOR / (A: DERIV;K: INTEGER) RES: DERIV;
  BEGIN
    A.X := A.X/K;
    A.PRIME := A.PRIME/K;
    RES := A
  END;

OPERATOR / (R: REAL;A: DERIV) RES: DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := R/A.X;
    U.PRIME := -A.PRIME*U.X/A.X;
    RES := U;
  END;

OPERATOR / (A: DERIV;R: REAL) RES: DERIV;
  BEGIN
    A.X := A.X/R;
    A.PRIME := A.PRIME/R;
    RES := A
  END;

OPERATOR / (A,B: DERIV) RES: DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := A.X/B.X;
    U.PRIME := (A.PRIME - U.X*B.PRIME)/B.X;
    RES := U
  END;
```

Functions for type DERIV have names beginning with D and source code for them is given in the files *.FUN, where * = DSQRT, DEXP, DLN, DSIN, DCOS, DARCTAN.

1. SQUARE ROOT

```
FUNCTION DSQRT(A: DERIV): DERIV;
  BEGIN
    A.X := SQRT(A.X);
    A.PRIME := 0.5*A.PRIME/A.X;
    DSQRT := A
  END;
```

2. EXPONENTIAL FUNCTION (base e)

```
FUNCTION DEXP(A: DERIV): DERIV;
  BEGIN
    A.X := EXP(A.X);
    A.PRIME := A.X*A.PRIME;
    DEXP := A
  END;
```

3. NATURAL LOGARITHM

```
FUNCTION DLN(A: DERIV): DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := LN(A.X);
    U.PRIME := A.PRIME/A.X;
    DLN := U
  END;
```

4. SINE

```
FUNCTION DSIN(A: DERIV): DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := SIN(A.X);
    U.PRIME := A.PRIME*COS(A.X);
    DSIN := U
  END;
```

5. COSINE

```
FUNCTION DCOS(A: DERIV): DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := COS(A.X);
    U.PRIME := -A.PRIME*SIN(A.X);
    DCOS := U
  END;
```

6. ARCTANGENT

```
FUNCTION DARCTAN(A: DERIV): DERIV;
    VAR U: DERIV;
  BEGIN
    U.X := ARCTAN(A.X);
    U.PRIME := A.PRIME/(1.0 + A.X*A.X);
    DARCTAN := U
  END;
```

APPENDIX C

A Pascal-SC Program for Newton's Method

```
PROGRAM LITTLEN(INPUT,OUTPUT);

    (* This program applies Newton's method to find a zero of a function
       f(x) of one variable using differentiation arithmetic. *)

   TYPE DERIV = RECORD X,PRIME: REAL END;

    (* If F is of type DERIV, then F.X = F(X), F.PRIME = F'(X). *)

    VAR X: DERIV;     (* THE INDEPENDENT VARIABLE *)
        F: DERIV;     (* THE DEPENDENT VARIABLE *)
        K: INTEGER;   (* ITERATION COUNTER *)
        C: CHAR;      (* CONTROL CHARACTER *)

    (* INSERT THE OPERATORS AND FUNCTIONS FOR TYPE DERIV YOU NEED TO
       EVALUATE YOUR FUNCTION HERE, FOR EXAMPLE: *)

$INCLUDE DADD.OPR;
$INCLUDE DSUB.OPR;
$INCLUDE DMUL.OPR;
$INCLUDE DDIV.OPR;

    (* END OF DERIV OPERATORS AND FUNCTIONS; THE ONES GIVEN ABOVE ARE
       ADEQUATE FOR ANY RATIONAL FUNCTION. *)

BEGIN  (* PROGRAM LITTLEN *)

    X.PRIME := 1.0;C := 'R';WHILE C = 'R' DO

    BEGIN  (* SET INITIAL VALUES *)

        K := 0;  (* SET ITERATION COUNTER TO 0 *)
        WRITELN;
        WRITELN('Enter initial value of X:');READ(X.X);
        C := 'I';WHILE C = 'I' DO

        BEGIN  (* NEWTON ITERATION *)

    (* INSERT A STATEMENT OR STATEMENTS TO EVALUATE F(X) HERE,
       FOR EXAMPLE: *)

            F := (X - 1)*(X + 3)/(X + 2);

    (* END OF DEFINITION OF F(X) *)
```

```
            WRITELN;  (* OUTPUT CURRENT VALUES OF X, F(X), AND F'(X) *)
            WRITELN('VALUES AT ITERATION NUMBER ',K:3,' ARE:');
            WRITELN;
            WRITELN('    X = ',X.X);
            WRITELN(' F(X) = ',F.X);
            WRITELN('F''(X) = ',F.PRIME);
            WRITELN;  (* END OUTPUT OF CURRENT VALUES *)

            WRITELN('ENTER "I" TO ITERATE, "R" TO RESTART, "Q" TO QUIT');
            READ(C,C);  (* The first character read from the terminal is
                           always a ' '. *)

            IF C = 'I' THEN
            BEGIN  (* NEWTON STEP *)

                X.X := X.X - F.X/F.PRIME;  (* NEW VALUE OF X *)
                  K := K + 1               (* INCREASE ITERATION COUNTER *)

            END;   (* NEWTON STEP *)

        END;   (* NEWTON ITERATION *)

    END;  (* COMPUTATION WITH GIVEN INITIAL VALUES *)

END.   (* PROGRAM LITTLEN *)
```

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 2688	2. GOVT ACCESSION NO. AD A142 764	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE ARITHMETIC OF DIFFERENTIATION		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 53706		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 May 1984 13. NUMBER OF PAGES 28
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

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

This report describes automatic differentiation, which is neither symbolic nor approximate, for single functions of one real variable. The rules of evaluation and differentiation are combined into an ordered-pair arithmetic similar to complex arithmetic, but slightly simpler. Evaluation of the formula for a function in this arithmetic yields both the values of (cont.)

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (cont.)

the function and its derivative, without a formula for the derivative of the function, and without numerical approximation, since this arithmetic is based on the well-known rules for differentiation. The properties of this arithmetic are examined, and illustrated by simple examples. Subroutines are given for differentiation arithmetic both on a hand-held programmable calculator, and in the microcomputer language Pascal-SC. An application of this arithmetic to the solution of equations by Newton's method is given, using a Pascal-SC program.