AECL-10139

ATOMIC ENERGY OF CANADA LIMITED

ÉNERGIE ATOMIQUE DU CANADA LIMITÉE

AUTOMATIC DIFFERENTIATION OF FUNCTIONS

DIFFÉRENTIATION AUTOMATIQUE DES FONCTIONS

S.R. DOUGLAS

Chalk River Nuclear Laboratories

Laboratoires nucléaires de Chalk River

Chalk River, Ontario K0J 1J0

June 1990 juin

ATOMIC ENERGY OF CANADA LIMITED

AUTOMATIC DIFFERENTIATION OF FUNCTIONS

by

S.R. DOUGLAS

Mathematics & Computation Branch
Chalk River Laboratories
Chalk River, Ontario K0J 1J0
1990 June

AECL-10139

ÉNERGIE ATOMIQUE DU CANADA LIMITÉE

DIFFÉRENTIATION AUTOMATIQUE DES FONCTIONS

par

S.R. DOUGLAS

RÉSUMÉ

La différentiation automatique est une méthode de calcul des dérivées de fonctions à n'importe quel ordre pour n'importe quel nombre de variables. Les fonctions doivent pouvoir être exprimées sous forme de combinaison de fonctions élémentaires. Lorsqu'évaluées à certaines valeurs numériques particulières, les dérivées n'ont pas d'erreur de troncation et sont obtenues automatiquement. La méthode est illustrée à l'aide d'exemples simples. Le programme source en FORTRAN est fourni.

Mathématiques et calculs
Laboratoires de Chalk River
Chalk River (Ontario) K0J 1J0
Juin 1990

EACL-10139

ATOMIC ENERGY OF CANADA LIMITED

AUTOMATIC DIFFERENTIATION OF FUNCTIONS

by

S.R. DOUGLAS

ABSTRACT

Automatic differentiation is a method of computing derivatives of functions to any order in any number of variables. The functions must be expressible as combinations of elementary functions. When evaluated at specific numerical points, the derivatives have no truncation error and are automatically found. The method is illustrated by simple examples. Source code in FORTRAN is provided.

Mathematics & Computation Branch
Chalk River Laboratories
Chalk River, Ontario K0J 1J0
1990 June

AECL-10139

INTRODUCTION

From one's first exposure to the calculus, the act of finding the derivative of a function is seen to be a fundamental process in mathematics. In class, the tutor torments the student by having him or her differentiate increasingly more complicated functions, first of one independent variable and later of several variables. Sometimes the student wishes that there was a better way, a way that would automatically calculate the intricate combinations of chain rules, product rules, and elementary derivatives, all the while holding some variables fixed and permitting others to vary.

Those who survive this ordeal and become scientists, engineers, or mathematicians have found that powerful computer codes such as REDUCE, MACSYMA, or MATHEMATICA can provide analytical formulas for derivatives of functions on demand, although the resultant formulas often extend over very many pages of output and take considerable time to express. If the user actually needs to know the values of these derivatives at specific values of the independent variables (often called points), the computer algebra codes will produce FORTRAN programs to evaluate these expressions numerically. As any practitioner of the analytical differentiation method knows, the repeated evaluation of these complicated analytical formulas can be very time consuming, even on the fastest computer.

Another approach to differentiation returns to the definition of derivative as a limiting process and approximates the value of the derivative at a specific point by differences in the function values at finite distances from this point. This is the finite difference method for numerically approximating the exact derivative at a specific point. The accuracy of this numerical differentiation depends both on the distance from the specified point to the points of function evaluation and on the sophistication of the formula that uses the function values. Any discrepancy between the approximate estimate and the actual value of the derivative is dubbed the truncation error. Better accuracy requires substantially more function evaluations or more complicated formulas, both of which limit the utility of numerical differentiation in many instances. Practitioners of the numerical differentiation method know that higher derivatives are very hard to get accurately.

There is a third approach to differentiation. It is not as widely known as the other two approaches and certainly not as widely used. But this third approach--automatic differentiation--has many advantages and, consequently, many applications in which it is preferred.

This paper describes a general algorithm for automatic differentiation that I developed. It is based on an approach suggested by Berz [14]. Before we can see how it fits into the field of automatic differentiation, we should see exactly what automatic differentiation is.

WHAT IS AUTOMATIC DIFFERENTIATION?

A simple example will illustrate. We are required to find the function value and the first derivative for the function f(x) = x ** 5 of the single variable x when x = 3. Of course, this example is trivial by analytical methods and straightforward by numerical methods.

To do this simple example by automatic differentiation, we first define a set of ordered pairs (A1,A2) with A1 and A2 real numbers. Given any three ordered pairs A=(A1,A2), B=(B1,B2), and C=(C1,C2), then addition is defined as C = A + B with (C1,C2) = (A1+B1,A2+B2) componentwise. This is a natural way to define addition. For multiplication we define C = A * B as (C1,C2) = (A1*B1,A1*B2+A2*B1). At first glance, this is not a natural way to define multiplication.

Returning to our original problem of function evaluation, we replace x with X = (3,1), f(x) with F = (F1,F2) and multiplication with our new definition of multiplication. Thus:

```
      F = X * X * X * X * X
or
     (F1,F2) = (3,1) * (3,1) * (3,1) * (3,1) * (3,1)
             = (9,6)         * (3,1) * (3,1) * (3,1)
             = (27,27)               * (3,1) * (3,1)
             = (81,108)                      * (3,1)
             = (243,405)
```

Analytically, f(3) = 243 and f'(3) = 405. For this simple example we see that our prescription has given the correct function value and the correct first derivative.

Several points should be noted. First, in accord with the wishes of our hapless student, we do not need to know anything about the rules of differentiation to evaluate the derivative. Second, as in numerical differentiation, the prescription yields numerical values at specific numerical points. Unlike numerical differentiation, no truncation error occurs. Also, the product rule is automatically built into the prescription. Finally, although it is not apparent from this simple example, this prescription is not as complex as analytical derivatives for more challenging problems. In some sense, automatic differentiation lies midway between analytical and numerical differentiation: less complicated than analytic; results evaluated at specific points; no truncation error; no rules of calculus to be remembered.

Consider a second example. Let f(x,y) = (x + 2 * y) ** 2 with x = 3 and y = 4. We want both first and second partial derivatives. Define a multiplet A = (A1,A2,A3,A4,A5,A6) with real components. Define addition

componentwise, as before. Define multiplication by the scalar 2 as componentwise multiplication. Define multiplet multiplication as follows:

```
C = A * B  as
             C1 = A1 * B1
             C2 = A1 * B2 + A2 * B1
             C3 = A1 * B3 + A3 * B1
             C4 = A1 * B4 + A4 * B1 + A2 * B2
             C5 = A1 * B5 + A5 * B1 + A2 * B3 + A3 * B2
             C6 = A1 * B6 + A6 * B1 + A3 * B3
```

Now replace x with X = (3,1,0,0,0,0) and y with Y = (4,0,1,0,0,0) and evaluate:

```
(F1,F2,F3,F4,F5,F6) = ( (3,1,0,0,0,0)+2*(4,0,1,0,0,0) ) ** 2
                    = (11,1,2,0,0,0) * (11,1,2,0,0,0)
                    = (121,22,44,1,4,4)
```

Analytically, f(3,4)=121, df(3,4)/dx=22, df(3,4)/dy=44, d2f(3,4)/dxdx=2, d2f(3,4)/dxdy=4, and d2f(3,4)/dydy=8. These results do not seem to agree until one realizes that F1,F2,F3,F4,F5,F6 should be interpreted as the terms in a Taylor series expansion of f(x,y) about (3,4). Then F4 has 2! built into it, F5 has 1!1! built into it, and F6 has 2! built into it. The exact results are then correctly produced.

As well as the points noted after the first example, we observe that more than one independent variable can be handled, the chain rule for composition of mappings is built into the prescription and, most importantly, the definition of multiplication has changed.

This prescription for automatic differentiation would be an amusing curiosity if it could only be used for multivariable polynomials. To provide a truly useful scheme for automatic differentiation, we need to be able to do transcendental functions. Our next example considers the natural logarithm of x.

Let f(x) = log(x) with single variable x = 2. We are required to find up to fourth derivatives using automatic differentiation. First we define our multiplication. Let A=(A1,A2,A3,A4,A5), B=(B1,B2,B3,B4,B5) and C=(C1,C2,C3,C4,C5). Let C = A * B as follows:

```
C1 = A1 * B1
C2 = A1 * B2 + A2 * B1
C3 = A1 * B3 + A3 * B1 + A2 * B2
C4 = A1 * B4 + A4 * B1 + A2 * B3 + A3 * B2
C5 = A1 * B5 + A5 * B1 + A2 * B4 + A4 * B2 + A3 * B3
```

Set X = (X1,X2,X3,X4,X5) = (2,1,0,0,0). Now we need an algorithm for evaluating log(X). Consider the following manipulations:

```
log(X) = log( (X1,X2,X3,X4,X5) )
       = log( (X1,0,0,0,0) + (0,X2,X3,X4,X5) )
       = log( X1 * ( (1,0,0,0,0) + (0,X2/X1,X3/X1,X4/X1,X5/X1) ) )
       = log(X1) + log(I+Z) with X1 not 0, I=(1,0,0,0,0) and
     Z = (0,X2/X1,X3/X1,X4/X1,X5/X1).
```

The first term is just the evaluation of the natural logarithm of the scalar X1 times (1,0,0,0,0) or (log(X1), 0, 0, 0, 0). The second term can be expanded in the infinite series log(I+Z)=Z-Z**2/2+Z**3/3-Z**4/4+Z**5/5 -... Now notice one crucial and remarkable fact. With any Z=(0,Z2,Z3,Z4,Z5), Z**5 = (0,0,0,0,0), Z**6 = Z*Z**5 = (0,0,0,0,0), etc. This means we need evaluate only a finite number of terms in our expansion! For any Z so specified, log(I+Z) = Z-Z**2/2+Z**3/3-Z**4/4 exactly.

Returning to our specific X = (2,1,0,0,0), then

```
(F1,F2,F3,F4,F5) = (log(2),0,0,0,0)
                 + (0,1/2,0,0,0)
                 - (0,1/2,0,0,0)*(0,1/2,0,0,0) /2
                 + (0,1/2,0,0,0)*(0,1/2,0,0,0)*(0,1/2,0,0,0) /3
                 - (0,1/2,0,0,0)*(0,1/2,0,0,0)*(0,1/2,0,0,0)*(0,1/2,0,0,0) /4
                 = ( log(2), 1/2, -1/8, 1/24, -1/64 )
```

The derivatives of log(x) at x = 2 are correctly given as 1/2*1!, -1/8*2!, 1/24*3!, -1/64*4!. Therefore, we have managed to do a transcendental function, the derivatives are exact, a small number of manipulations are required, and no new transcendental functions need to be evaluated. Quite remarkable.

HOW AUTOMATIC DIFFERENTIATION WORKS

Although the three worked examples have shown that something interesting occurs with the use of our prescription, the reader must be left with more questions than answers. Some of these questions, among others, may be: why does automatic differentiation work?; what is the appropriate definition of multiplication for v variables and nth order derivatives?; when can we derive successful formulas for other transcendental functions?; what are the meanings of the ones in the multiplets for independent variables?; how many terms are there in a multiplet for v independent variables and nth order derivatives?; what use is this stuff, anyway? We attempt to answer some of these questions below.

(a) Deriving Formulas

When can we derive successful formulas for other functions? This question actually has two parts: (1) when can we derive a formula? and (2) when can we code this formula efficiently and generally in a computer program? We can derive a formula for a given one-variable function when

the function has a convergent power series (i.e., is analytic) within some region of the specified value of the independent variable. The restriction to one-variable functions is not a restriction at all because a change of variable can turn any multivariable elementary function into a one-variable function. Automatic differentiation would keep track of the hidden partial derivatives in such a change of variable. As for the question of efficient coding, I do not know the full answer. Nevertheless, I have managed to code general routines for the intrinsic FORTRAN functions. Each function needs some ingenuity and uses a slightly different trick. One could spend one's career coding more and more special functions but I have decided to code only those needed by specific applications.

(b) Number Of Terms In Multiplet

How many terms are there in a given multiplet? First, some notation. Let nDv be the set of multiplets together with the appropriate multiplication for multiplets defined to give nth order derivatives with v independent variables. The definition of multiplication differs with each n and v. Let N(n,v) be the number of terms in a multiplet. Each term in a given multiplet represents a coefficient in the Taylor series expansion of a function evaluated at a specific point. Each term is therefore a product (often called a monomial) of some of the v independent variables. The question, posed differently, is: How many monomials are there in v variables up to and including nth powers? The number of terms N(n,v) can be divided into two categories: those monomials that contain the last variable and those that do not. For those monomials that do not contain the last variable, there are evidently N(n,v-1) of them. For those monomials that do, factor out the last variable and you are left with (n-1)Dv, so there are N(n-1,v) terms. Consequently,

$$N(n,v) = N(n,v-1) + N(n-1,v).$$

For example, in 3D4, with w the "last variable", the monomials are:

```
{1,x,y,z,w,x*x,x*y,x*z,x*w,y*y,y*z,y*w,z*z,z*w,w*w,x*x*x,x*x*y,
 x*x*z,x*x*w,x*y*y,x*y*z,x*y*w,x*z*z,x*z*w,x*w*w,y*y*y,y*y*z,
 y*y*w,y*z*z,y*z*w,y*w*w,z*z*z,z*z*w,z*w*w,w*w*w}

= {1,x,y,z,x*x,x*y,x*z,y*y,y*z,z*z,x*x*x,x*x*y,x*x*z,x*y*y,x*y*z,
   x*z*z,y*y*y,y*y*z,y*z*z,z*z*z}
 +w*{1,x,y,z,w,x*x,x*y,x*z,x*w,y*y,y*z,y*w,z*z,z*w,w*w}
```

So N(n,v) = N(3,4) = 35, N(n,v-1) = N(3,3) = 20, N(n-1,v) = N(2,4) = 15 and N(n,v) = N(n,v-1) + N(n-1,v).

We have a two-variable recurrence relation on N(n,v) that is to be solved. Let Gn(x) = N(n,0)*1 + N(n,1)*x + N(n,2)*x**2 + N(n,3)*x**3 + ... be a combinatorial generating function. Using the recurrence relation on each term of Gn(x)-N(n,0) yields Gn(x)-N(n,0) = x*Gn(x) + Gn-1(x) - N(n-1,0). With boundary conditions of N(n,0)=1 for n>0 and N(0,v)=1 for v>0, the result becomes Gn(x) = Gn-1(x) / (1-x) = G0(x) / (1-x)**n = 1 / (1-x)**(n+1). From the binomial expansion for 1/(1-x)**(n+1), each term is compared with the terms of the generating function. The result is:

$$N(n,v) = (n+v)! / n! / v!.$$

(c) Definition Of Multiplication

What is the appropriate definition of multiplication? To define multiplication C = A * B of two multiplets of length N(n,v), we need "only" find all pairs of monomials that can make up a given monomial. The corresponding term in the multiplet is used in the definition. For example, in 3D4 above, the 24th term x*z*w can be made up of (1)*(x*z*w),(x*z*w)*(1), (x)*(z*w),(z*w)*(x),(z)*(x*w),(x*w)*(z),(w)*(x*z),(x*z)*(w). Therefore, in the definition of multiplication, C24 = A1 * B24 + A24 * B1 + A2 * B14 + A14 * B2 + A4 * B9 + A9 * B4 + A5 * B8 + A8 * B5. A general formula for arbitrary n and v will be exhibited later in this paper.

(d) How It Works

Now for the really tough question: Why does automatic differentiation work? From a mathematical point of view, the multiplets together with componentwise addition and multiplet multiplication form a mathematical system known as a ring. A ring has the following properties:

(1) the addition of any two elements of a ring is a third element of the ring (closure);
(2) addition is associative, i.e., (A + B) + C = A + (B + C);
(3) a neutral element 0 (called zero) exists such that A + 0 = A for all A. Here 0=(0,0,0,0,...,0);
(4) every element A has another element B in the ring as its additive inverse, i.e., A + B = 0;
(5) the order of addition is irrelevant (commutative), i.e., A+B = B+A.

These properties make the set of multiplets an Abelian group under addition. Furthermore,

(6) the product of any two elements of a ring is a third element of the ring (closure);
(7) multiplication is associative, i.e., (A * B) * C = A * (B * C).

These seven properties are required for any mathematical system to be a ring. A given ring such as our multiplet system may or may not have other properties of interest. Ours also has the properties that:

(8) I=(1,0,0,0,...,0) gives I*A=A*I=A for all A. It is an identity element under multiplication;
(9) A * B = B * A for any pair of elements (commutative);
(10) multiplication is distributive over addition, i.e., A*(B+C)=A*B+A*C.

Rings may not be foreign territory to the reader because the integer numbers form a ring under addition and multiplication. As well as the above ten properties, the ring of integers also has the property that i * j = 0 implies either i = 0 or j = 0. But our ring does not have this property! Consider A=(0,0,0,0,...,0,1) and B=(0,0,0,0,...,1,0) with any appropriate multiplication rule. In the jargon, our ring is not an integral domain.

Another interesting property exhibited by the ring of rational numbers is division: for every non-zero q, there is an r in the rationals such that q * r = 1. Neither the integers nor our ring enjoys this property. Consider A=(0,A2,A3,A4,...,AN) with any appropriate multiplication rule.

The selection of elements of the general form A=(0,A2,A3,A4,...,AN) leads to another important concept in ring theory. A set U of a ring is called an ideal if U is a subgroup under addition and if any element of the ring times any element of U is an element of U. Prosaically, U "swallows up" multiplication by arbitrary ring elements. This is precisely what the set in the form of A above does. Furthermore, so does B=(0,0,B3,B4,...,BN). And so does C=(0,0,0,C4,...,CN), and so on. Careful examination of the multiplication rule shows that all these sets are ideals. Consequently, each ring with a given multiplication rule has (N-1) ideals. Along with these is the set R=(R1,0,0,0,...,0), which is essentially the real numbers. Any arbitrary element of the ring can decomposed into a sum of elements from R and the (N-1) ideals, labelled Ui, i=2,N. For example,

X=(X1,X2,X3,X4,...,XN) = (X1,0,0,0,...,0) + (0,X2,0,0,...,0) +
(0,0,X3,0,...,0) + (0,0,0,X4,0,...,0) + ... + (0,0,0,0,...,0,XN).

By definition, we already know that any X times an element of the ideal Ui is an element of Ui. But under what conditions does Ui * Uj decompose into a smaller ideal Uk with k>i and k>j? The answer: always. This is not a general property of rings but a specific property of our ring, due to our multiplication rule. Incidently, U2=(0,A2,A3,A4,...,AN) is known as a maximal ideal. All of the other ideals are subsets of U2. The fact that in our case the product of two ideals is always a smaller ideal means that any element of U2 times itself repeatedly will eventually be zero. All ideals are nilpotent of order at most (n+1). We need only keep n terms (a comparatively small number) in our series expansions of transcendental functions.

(e) The Meaning Of The Ones

What are the meanings of the ones in the multiplets for independent variables? Consider a specific example of three independent variables (v=3). Up to second-order derivatives (n=2) there will be ten terms (N=10) in a multiplet. The element X=(x,1,0,0,0,0,0,0,0,0) selects the first ideal and associates it with the x-variable. The element Y=(y,0,1,0,0,0,0,0,0,0) selects the second ideal and associates it with the y-variable. The element Z=(z,0,0,1,0,0,0,0,0,0) selects the third ideal and associates it with the z-variable. The value 1 represents the derivative of the independent variable with respect to itself. The multiplication rule and its corresponding set and structure of ideals then keeps track of the requisite relationships.

To complete the abstract discussion, the set of multiplets also forms a vector space over the real numbers. With certain minor restrictions, a ring that is also a vector space forms a mathematical system known as an "algebra". Consequently, the topic of automatic differentiation is sometimes called Differential Algebra.

OTHER EXAMPLES

With the abstract discussion now completed, we can proceed to the easier task of giving further examples of the automatic differentiation of functions.

To calculate division (with X1 not 0), we proceed as follows:

```
1 / (X1,X2,...,XN) = 1/X1 / ( (1,0,...,0) + (0,X2/X1,X3/X1,...,XN/X1) )
                   = (      (1,0,...,0)
                     -      (0,X2/X1,X3/X1,...,XN/X1)
                     +      (0,X2/X1,X3/X1,...,XN/X1) ** 2
                     -      (0,X2/X1,X3/X1,...,XN/X1) ** 3
                      ...
                   (-1)**n (0,X2/X1,X3/X1,...,XN/X1) ** n ) / X1.
```

We know that there are a small number of terms in the series because some power of the maximal ideal is zero.

To calculate a sine function, we proceed as follows:

```
sin( (X1,X2,...,XN) )
   = sin( (X1,0,0,...,0) + (0,X2,X3,...,XN) )
   = sin( (X1,0,0,...,0) ) * cos( (0,X2,X3,...,XN) )
     + cos( (X1,0,0,...,0) ) * sin( (0,X2,X3,...,XN) )
   = sin(X1) * ( (1,0,0,...,) - (0,X2,X3,...,XN) ** 2 / 2!
                 + (0,X2,X3,...,XN) ** 4 / 4! - ... (to nth power ) )
    +cos(X1) * ( (0,X2,X3,...,XN) - (0,X2,X3,...,XN) ** 3 / 3!
                 + (0,X2,X3,...,XN) ** 5 / 5! - ... (to nth power ) ).
```

To calculate the cosine function is essentially the same.

A list of functions that I have programmed in this manner is:

1/x, exp(x), log(x), sqrt(x), cos(x), sin(x), tan(x), cosh(x), sinh(x), tanh(x), arccosh(x), arcsinh(x), arctanh(x), erf(x).

Others could be programmed as required. The inverse trigonometric functions, however, have resisted attempts to find a general algorithm. Compositions of any of the above functions are also available. The variable x represents any number of variables.

As an aside, how does one test whether the algorithms are properly programmed? The problem arises when the number of derivatives requested is large. First, the algorithms were checked against known analytical cases such as exp(x) and sin(x), which are easy to do. Second, the computer algebra code REDUCE was used to generate more complicated analytical derivatives, which were then compared at specific points with the automatic differentiation algorithms. But the best way that I have found is to evaluate a functional identity such as sin(x)**2 + cos(x)**2 = 1 and observe that all higher derivatives manipulate to zeros. Several functional identities were used to check the programming of the algorithms.

THE GENERAL FORMULA

Some implementations of automatic differentiation create tree data structures to establish the correct definition of multiplication. The coding for these implementations tends to be complicated because of the significant effort needed to create and maintain a dynamic data structure for trees. Compounding this complexity is the fact that most scientific programming is still done in FORTRAN, a poor language for the implementation of dynamic data structures.

The approach I have taken defines multiplication explicitly at the beginning of the calculation. I found this difficult to do until I hit upon the idea of transforming the problem to a new, equivalent problem. A general algorithm for multiplication could then be formulated.

The objective is to predict which pair of monomials can be multiplied together to produce a given monomial. We must have the algorithm do this for all monomials with any order of derivative in any number of variables. The result is coded as a pair of integer indices representing the two ring ideals that make up the product ring ideal with its associated integer index. For example, in five variables (x,y,z,u,w), consider the target monomial x**2 * y**3 * z. The exponents also form the quintuple (2,3,1,0,0). The idea is to transform the quintuple of exponents to the following form: [2+3+1+0+0,3+1+0+0,1+0+0,0+0,0] = [6,4,1,0,0]. The virtue of the transformed quintuple is that we can readily find what integer index the ring ideal corresponds to by using the counting formula N(n,v) = (n+v)! / n! / v!.

An example, although a bit messy, will serve to elucidate the structure of the general formula for multiplication. Consider five variables (x,y,z,u,w) with derivatives up to third order. Therefore, v = 5, n = 3, N(n,v) = 56. A listing of the monomials, their exponent quintuples, and the transformed exponent quintuples is:

Monomial		Exponent Quintuplet		Transformed Quintuplet	Comments
1	-->	(0 0 0 0 0)	-->	[0 0 0 0 0]	
					subtotal = 1
x	-->	(1 0 0 0 0)	-->	[1 0 0 0 0]	
y	-->	(0 1 0 0 0)	-->	[1 1 0 0 0]	
z	-->	(0 0 1 0 0)	-->	[1 1 1 0 0]	
u	-->	(0 0 0 1 0)	-->	[1 1 1 1 0]	
w	-->	(0 0 0 0 1)	-->	[1 1 1 1 1]	
					subtotal = 6
x*x	-->	(2 0 0 0 0)	-->	[2 0 0 0 0]	
x*y	-->	(1 1 0 0 0)	-->	[2 1 0 0 0]	
x*z	-->	(1 0 1 0 0)	-->	[2 1 1 0 0]	
x*u	-->	(1 0 0 1 0)	-->	[2 1 1 1 0]	
x*w	-->	(1 0 0 0 1)	-->	[2 1 1 1 1]	

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    y*y       -->  ( 0 2 0 0 0 )  -->  [ 2 2 0 0 0 ]
    y*z       -->  ( 0 1 1 0 0 )  -->  [ 2 2 1 0 0 ]
    y*u       -->  ( 0 1 0 1 0 )  -->  [ 2 2 1 1 0 ]
    y*w       -->  ( 0 1 0 0 1 )  -->  [ 2 2 1 1 1 ]
    z*z       -->  ( 0 0 2 0 0 )  -->  [ 2 2 2 0 0 ]
    z*u       -->  ( 0 0 1 1 0 )  -->  [ 2 2 2 1 0 ]
    z*w       -->  ( 0 0 1 0 1 )  -->  [ 2 2 2 1 1 ]
    u*u       -->  ( 0 0 0 2 0 )  -->  [ 2 2 2 2 0 ]
    u*w       -->  ( 0 0 0 1 1 )  -->  [ 2 2 2 2 1 ]
    w*w       -->  ( 0 0 0 0 2 )  -->  [ 2 2 2 2 2 ]
_______________________________________________________________  subtotal = 21
    x*x*x     -->  ( 3 0 0 0 0 )  -->  [ 3 0 0 0 0 ]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -     subsubtotal=1
    x*x*y     -->  ( 2 1 0 0 0 )  -->  [ 3 1 0 0 0 ]
    x*x*z     -->  ( 2 0 1 0 0 )  -->  [ 3 1 1 0 0 ]
    x*x*u     -->  ( 2 0 0 1 0 )  -->  [ 3 1 1 1 0 ]
    x*x*w     -->  ( 2 0 0 0 1 )  -->  [ 3 1 1 1 1 ]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -     subsubtotal=5
    x*y*y     -->  ( 1 2 0 0 0 )  -->  [ 3 2 0 0 0 ]
    x*y*z     -->  ( 1 1 1 0 0 )  -->  [ 3 2 1 0 0 ]
    x*y*u     -->  ( 1 1 0 1 0 )  -->  [ 3 2 1 1 0 ]
    x*y*w     -->  ( 1 1 0 0 1 )  -->  [ 3 2 1 1 1 ]
    x*z*z     -->  ( 1 0 2 0 0 )  -->  [ 3 2 2 0 0 ]
    x*z*u     -->  ( 1 0 1 1 0 )  -->  [ 3 2 2 1 0 ]
    x*z*w     -->  ( 1 0 1 0 1 )  -->  [ 3 2 2 1 1 ]
    x*u*u     -->  ( 1 0 0 2 0 )  -->  [ 3 2 2 2 0 ]
    x*u*w     -->  ( 1 0 0 1 1 )  -->  [ 3 2 2 2 1 ]
    x*w*w     -->  ( 1 0 0 0 2 )  -->  [ 3 2 2 2 2 ]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -     subsubtotal=15
    y*y*y     -->  ( 0 3 0 0 0 )  -->  [ 3 3 0 0 0 ]
    y*y*z     -->  ( 0 2 1 0 0 )  -->  [ 3 3 1 0 0 ]
    y*y*u     -->  ( 0 2 0 1 0 )  -->  [ 3 3 1 1 0 ]
    y*y*w     -->  ( 0 2 0 0 1 )  -->  [ 3 3 1 1 1 ]
    y*z*z     -->  ( 0 1 2 0 0 )  -->  [ 3 3 2 0 0 ]
    y*z*u     -->  ( 0 1 1 1 0 )  -->  [ 3 3 2 1 0 ]
    y*z*w     -->  ( 0 1 1 0 1 )  -->  [ 3 3 2 1 1 ]
    y*u*u     -->  ( 0 1 0 2 0 )  -->  [ 3 3 2 2 0 ]
    y*u*w     -->  ( 0 1 0 1 1 )  -->  [ 3 3 2 2 1 ]
    y*w*w     -->  ( 0 1 0 0 2 )  -->  [ 3 3 2 2 2 ]
    z*z*z     -->  ( 0 0 3 0 0 )  -->  [ 3 3 3 0 0 ]
    z*z*u     -->  ( 0 0 2 1 0 )  -->  [ 3 3 3 1 0 ]
    z*z*w     -->  ( 0 0 2 0 1 )  -->  [ 3 3 3 1 1 ]
    z*u*u     -->  ( 0 0 1 2 0 )  -->  [ 3 3 3 2 0 ]
    z*u*w     -->  ( 0 0 1 1 1 )  -->  [ 3 3 3 2 1 ]
    z*w*w     -->  ( 0 0 1 0 2 )  -->  [ 3 3 3 2 2 ]
    u*u*u     -->  ( 0 0 0 3 0 )  -->  [ 3 3 3 3 0 ]
    u*u*w     -->  ( 0 0 0 2 1 )  -->  [ 3 3 3 3 1 ]
    u*w*w     -->  ( 0 0 0 1 2 )  -->  [ 3 3 3 3 2 ]
    w*w*w     -->  ( 0 0 0 0 3 )  -->  [ 3 3 3 3 3 ]                subsubtotal=35
_______________________________________________________________  subtotal = 56
```

There is evidently an underlying structure, particularly with the transformed exponent multiplets. We may see what this structure is by using the counting formula in tabular form.

Table of N(n,v) with n and v.

v \ n	0	1	2	3	...	n
0	1	1	1	1		
1	1	2	3	4		
2	1	3	6	10		
3	1	4	10	20		
4	1	5	15	35		
5	1	6	21	56		

Using this table to analyse the structure of the transformed exponent multiplets leads to the following conclusion. The cumulative subtotals of the number of multiplets up to the solid lines are:

1 (= N(0,5)), 6 (= N(1,5)), 21 (= N(2,5)), 56 (= N(3,5)).

The cumulative subtotals up to the dashed lines for each solid line are:

0
1, 5, 15
1 (= N(0,4)), 5 (= N(1,4)), 15 (= N(2,4)), 35 (= N(3,4)).

Similarly, the cumulative subtotals of the third variable z within the structure of the second variable y is:

1 (= N(0,3)), 4 (= N(1,3)), 10 (= N(2,3)), 20 (= N(3,3)).

And so on, recursively.

In general, given any transformed exponent multiplet [j(1),j(2),...,j(v)], we find the index of the corresponding monomial as

index = 1 + N(j(1)-1,v) + N(j(2)-1,v-1) + N(j(3)-1,v-2) + ... + N(j(v)-1,1)

but stopping the sum when j(k) = 0.

The construction of the ring multiplication is now straightforward. The product of two monomials makes a third monomial. Equivalently, the addition of two transformed exponent multiplets makes a third transformed exponent multiplet that corresponds to the third monomial. For example, in five variables, monomial x*y has multiplet [2 1 0 0 0] with index 8, monomial x*y*y*z has multiplet [4 3 1 0 0] with index 73, and the product monomial x*x*y*y*y*z has multiplet [6 4 1 0 0] with index 289.

Appendix 1 gives source code for the definition of multiplication. If your computer system does not permit extensible common in FORTRAN, then define a large block of common, testing the size required for a given application and informing the user that he must re-compile when he asks for too much. Also, the code assumes that all arrays and variables are preset to zero by the loader.

The definition of multiplication may now be used by invoking the times routine for multiplets C = A * B. I have been unable to vectorize this code. Appendix 2 gives source code to multiply two multiplets.

This paper provides little hint of the full power and scope of automatic differentiation. As a simple illustration, an exponential function in six variables (x1,x2,x3,x4,x5,x6) was evaluated up to ninth order in derivatives. It took less than one second on our mainframe (Cyber 990) and all 5005 derivative terms were accurate to at least twelve significant digits. The main feature of the implementation of automatic differentiation as described in this paper is the ease with which high-order derivatives can be computed in many variables.

IMPLEMENTATIONS BY OTHERS

Like many modern ideas, automatic differentiation has a long history. The first serious attempts to implement computer packages for differentiation seems to have occurred in the USSR [15] and the USA in the late 1950's. A large effort was made at the University of Wisconsin [29] and other places [16,30] in the mid 1960's. The advances made in automatic differentiation during and after this time are well documented in Rall's book [1]. Also, the epithet "automatic" was further strengthened by the writing of software to convert standard FORTRAN programs to include derivatives without the intervention of the user. There are at least five such pre-compilers available now for general use [14,19,20,24,27]. Several different methods of automatic differentiation have been proposed, each claiming certain advantages. As I see the situation, these can be divided into four classes: forward methods; reverse accumulation methods; code lists using graph theory; and ring multiplication methods. The implementation described in this paper is a forward ring method. I was first made aware of this topic from the recent work of Berz [13,14]. Much of the complexity of implementing automatic differentiation disappears when the programming language permits operator overloading, such as the language C++. However, the most likely user community is engineers and scientists who have a substantial investment in FORTRAN software. One mitigating solution may result from the possibility of operator overloading in FORTRAN 8X.

USES OF AUTOMATIC DIFFERENTIATION

Automatic differentiation can be used whenever derivatives are required. The technique has been used or could be used in the following fields of mathematics: root finding in nonlinear equations; constrained and unconstrained optimization; automatic error analysis; solution of ordinary and partial differential equations, including stiff equations and boundary value problems; maximum likelihood estimation; nonlinear least squares fitting; numerical integration; sensitivity analysis; chaos and nonlinear dynamics; stability analysis of differential equations; catastrophe theory; optimal control; integral equations; analysis of trajectories in Hamiltonian dynamics.

REFERENCES AND BIBLIOGRAPHY

1. L.B. Rall, Automatic Differentiation: Techniques And Applications, Springer-Verlag Lecture Notes In Computer Science, vol. 120, 1981. (This contains over 100 references, about half of which refer to early implementations of automatic differentiation.)
2. L.B. Rall, The Arithmetic Of Differentiation, Mathematics Magazine, vol. 59, no. 5, December 1986, pg. 275-282.
3. L.B. Rall, Differentiation In Pascal-SC: Type GRADIENT, ACM Trans. Math. Software, vol. 10, 1984, pg. 161-184.
4. L.B. Rall, Optimal Implementation Of Differentiation Arithmetic, Computer Arithmetic, Scientific Computation And Programming Languages, Teubner, 1987.
5. F.W. Pfeiffer, Automatic Differentiation In Prose, SIGNUM Newsletter, January 1987.
6. H. Kagiwada, R. Kalaba, N. Rasakhoo, and K. Springarn, Numerical Derivatives And Nonlinear Analysis, Plenum Press, 1986.
7. A. Griewank, On Automatic Differentiation, Mathematical Programming 88, Kluwer Academic Publishers, 1989.
8. V.L. Rvachev, G.P. Man'ko, V.V. Fed'ko, Technology Of Differentiating Functions Of Many Variables On An Electronic Computer, Kibernetika, no. 5, Sept-Oct 1983, pg. 31-33.
9. W. Baur and V. Strassen, The Complexity Of Partial Derivatives, Theoret. Comput. Sci., vol. 22, 1983, pg. 317-330.
10. M. Iri, Simultaneous Computation Of Functions, Partial Derivatives And Estimates Of Rounding Errors - Complexity And Practicality, Japan J. Appl. Math. vol. 1, 1984, pg. 223-252.
11. J. Joss, Algorithmisches Differenzieren, Ph.D. thesis, ETH, Zurich, Switzerland, 1976.
12. B. Speelpenning, Compiling Fast Partial Derivatives Of Functions Given By Algorithms, Ph.D. thesis, UIUCDS-R-80-1002, U. of Illinois at Urbana-Champaign, 1980.
13. M. Berz, Differential Algebraic Description And Analysis Of Trajectories In Vacuum Electronic Devices Including Space-Charge Effects, IEEE Transactions On Electronic Devices, vol. 35, no. 11, November 1988, pg. 2002-2009.
14. M. Berz, Differential Algebraic Description Of Beam Dynamics To Very High Orders, Particle Accelerators, vol. 24, 1989, pg. 109-124.
15. L.M. Beda et al., Programs For Automatic Differentiation For The Machine BESM, Inst. Precise Mechanics and Comp. Techniques, Moscow Academy Of Science, 1959.
16. R.E. Wengert, A Simple Automatic Derivative Evaluation Program, Com. ACM, vol. 7, 1964, pg. 463-464.
17. G. Kedem, Automatic Differentiation Of Computer Programs, ACM TOMS, vol. 6, no. 2, 1980, pg. 150-165.
18. H. Fisher, Automatic Differentiation: How To Compute The Hessian Matrix, Technical Report 104A, Technische Universitat Munchen, 1987.
19. M. Iri and K. Kubota, Methods Of Fast Automatic Differentiation And Applications, Research Memorandum RMI 87-0, Dept. Of Mathematical Engineering And Instrumentation Physics, U. Of Tokyo, 1987.
20. K.E. Hillstrom, Users Guide For JAKEF, Technical Memorandum ANL/MCS-TM-16, Mathematics And Computer Science Division, Argonne National Lab., 1985.

21. L.C.W. Dixon, Automatic Differentiation And Parallel Processing In Optimisation, Technical Report 180, The Hatfield Polytechnic, UK, 1987.
22. L.C.W. Dixon and M. Mohseninia, The Use Of The Extended Operations Set Of ADA With Automatic Differentiation And The Truncated Newton Method, Technical Report 176, The Hatfield Polytechnic, UK, 1987.
23. K.V. Kim et al., An Efficient Algorithm For Computing Derivatives And Extremal Problems, Ekonomika i matematicheskie metody, vol. 20, no. 2, 1984, pg. 309-318.
24. J.E. Horwedel et al., GRESS Version 0.0 Users Manual, ORNL/TM 10835, Oak Ridge National Laboratory, 1988.
25. P.H. Davis and J.D. Pryce, A New Implementation Of Automatic Differentiation For Use With Numerical Software, Technical Report AM-87-11, School Of Mathematics, U. Of Bristol, UK, 1987.
26. B.R. Stephens and J.D. Pryce, Passing Functions In FORTRAN For Automatic Differentiation, Technical Report AM-87-13, School Of Mathematics, U. Of Bristol, UK, 1987.
27. P.H. Davis et al., Specification Of A Preprocessor For Use With Differentiation Arithmetic, Technical Report AM-88-08, School Of Mathematics, U. Of Bristol, UK, 1988.
28. P.H. Davis et al., Bibliography On Methods And Techniques In Differentiation Arithmetic, Technical Report AM-88-09, School Of Mathematics, U. Of Bristol, UK, 1988.
29. R.E. Moore et al., DIFEQ Integration Routine - User's Manual, Technical Report LMSC 6-90-64-6, Lockheed Missiles And Space Company, 1964.
30. R.D. Wilkins, Investigation Of A New Analytical Method For Numerical Derivative Evaluation, Commumications ACM, vol. 7, 1964, pg. 465-471.

APPENDIX 1: SOURCE CODE FOR THE DEFINITION OF MULTIPLICATION

```
      INTEGER FUNCTION NTOTAL ( NORDER, NVAR, NBLOCKS, NLIST )
*
*
*  <NTOTAL> DOES SEVERAL THINGS.
*          1. CALCULATES THE NUMBER OF COMPONENTS IN A GIVEN ELEMENT OF THE
*             DIFFERENTIAL ALGEBRA.
*                      NTOTAL = N = ( NORDER + NVAR )! / NORDER! / NVAR!
*          2. CALCULATES THE GENERALIZED DEFINITION OF MULTIPLICATION FOR
*             THE DIFFERENTIAL ALGEBRA.
*               (a) FILL NBLOCKS WITH THE NUMBER OF TERMS FOR A FIXED NUMBER
*                   OF VARIABLES IVAR WITH IVAR = 0, NVAR.
*               (b) FILL NLIST WITH TRANSFORMED MULTIPLETS CONSISTING OF
*                   VARIOUS SUMS OF EXPONENTS OF MONOMIALS, E.G.
*                   X**2*Y**3*Z ==> (2,3,1,0,0) IN FIVE VARIABLES
*                               ==> [2+3+1+0+0,3+1+0+0,1+0+0,0+0,0]
*                                =  [6,4,1,0,0]
*                                =  NLIST(IVAR,*), IVAR = 1, NVAR.
*               (c) COMBINE PAIRS OF MONOMIALS TO CREATE NEW MONOMIALS. E.G.
*                   X*Y * X*Y**2*Z ==> (1,1,0,0,0) + (1,2,1,0,0) =(2,3,1,0,0)
*                                  <==> [2,1,0,0,0] + [4,3,1,0,0] = [6,4,1,0,0]
*               (d) BY USING AN ALGORITHM, DETERMINES THE COMPONENT INDEX
*                   MULTIPLET OF THE PRODUCT ELEMENT FOR EACH PAIR OF COMPONENT
*                   INDEX MULTIPLETS OF THE MONOMIAL ELEMENTS.
*                                    E.G. [2,1,0,0,0] <-->   8
*                                         [4,3,1,0,0] <-->  73
*                                         [6,4,1,0,0] <--> 289
*               (e) CONSEQUENTLY, THE DEFINITION OF MULTIPLICATION OF TWO
*                   ELEMENTS OF THE DIFFERENTIAL ALGEBRA A(N) AND B(N) TO
*                   PRODUCE A THIRD ELEMENT C(N) IS NOW DEFINED. E.G.
*                   C(289) = C(289) + A(8) * B(73) + A(73) * B(8) IS A TERM.
*
      INTEGER TOP, BOTTOM
      INTEGER NBLOCKS(0:NORDER,0:NVAR), NLIST(NVAR,*)
C$    EXTEND ( / TheRule / )
      COMMON / TheRule / ITRIPLE, NTRIPLE(3,1000)
*
*  CALCULATE NUMBER OF TERMS FOR ALL VARIABLES UP TO NVAR AND ALL ORDERS UP
*  TO NORDER
*
      DO 9 IORDER = 0, NORDER
    9   NBLOCKS(IORDER,0) = 1
```

```
*
      DO 2 IVAR = 1, NVAR
        NBLOCKS(0,IVAR) = 1
        DO 22 IORDER = 1, NORDER
          NSUM = IORDER + IVAR
          NLOOP = MIN ( IORDER, IVAR )
          TOP = 1
          BOTTOM = 1
          DO 1 I = 1, NLOOP
            TOP = TOP * ( NSUM + 1 - I )
            BOTTOM = BOTTOM * I
    1     CONTINUE
          NBLOCKS(IORDER,IVAR) = TOP / BOTTOM
   22   CONTINUE
    2 CONTINUE
      NTOTAL = NBLOCKS(NORDER,NVAR)
*
*  FILL NLIST WITH BASIC PATTERN FOR ALL ORDERS.
*
      DO 50 I = 1, NVAR
   50   NLIST(I,1) = 0
      ICOUNT = 1
      DO 3 IORDER =  0, NORDER
        DO 4 I = 1, NBLOCKS(IORDER,NVAR-1)
          NLIST(1,ICOUNT) = IORDER
    4   ICOUNT = ICOUNT + 1
    3 CONTINUE
*
      DO 65432 I = 1, ICOUNT-1
      DO 65432 IVAR = 2, NVAR
65432 NLIST(IVAR,I) = 0
*
*  USING BASIC PATTERN FOR ALL ORDERS, FILL PATTERN FOR ALL SUB-ORDERS OF
*  RESTRICTED SET OF VARIABLES IVAR = 2, NVAR.
*
      DO 60 IVAR = 2, NVAR
        DO 5 I = 2, NTOTAL
          IF ( NLIST(IVAR-1,I) .NE. NLIST(IVAR-1,I-1) ) THEN
             ICOUNT = I
               DO 6 IORDER = 0, NLIST(IVAR-1,I)
                 DO 7 J = 1, NBLOCKS(IORDER,NVAR-IVAR)
                   NLIST(IVAR,ICOUNT) = IORDER
    7            ICOUNT = ICOUNT + 1
    6          CONTINUE
          END IF
    5   CONTINUE
   60 CONTINUE
*
*  COMBINE PAIRS OF MONOMIALS TO PRODUCE THIRD MONOMIAL.
*  PAIRS THAT INVOLVE CONSTANTS ARE OBVIOUS, SO WE DO NOT NEED TO WASTE
*  COMPUTER TIME GENERATING THEM, E.G. C(M) = C(M) + A(1)*B(M) + A(M)*B(1).
```

```
*
      ITRIPLE = 0
      NSTOP = NORDER/2 + MOD(NORDER,2)
      MSTOP = NORDER - NSTOP
      DO 90 I = 2, NBLOCKS(MSTOP,NVAR)
        DO 85 J = I, NBLOCKS(NORDER-1,NVAR)
          IORDER = NLIST(1,I) + NLIST(1,J)
          IF ( IORDER .LE. NORDER ) THEN
             L = 1
             DO 44 K = 1, NVAR
               LTEST = NLIST(K,I) + NLIST(K,J)
               IF ( LTEST .NE. 0 ) THEN
                  L = L + NBLOCKS(LTEST-1,NVAR+1-K)
               END IF
   44        CONTINUE
             ITRIPLE = ITRIPLE + 1
             NTRIPLE(1,ITRIPLE) = I
             NTRIPLE(2,ITRIPLE) = J
             NTRIPLE(3,ITRIPLE) = L
          END IF
   85   CONTINUE
   90 CONTINUE
*
      RETURN
      END
```

APPENDIX 2: MULTIPLYING TWO MULTIPLETS TOGETHER

```
      SUBROUTINE TIMES ( N, NORDER, A, B, C )
      REAL A(N), B(N), C(N)
C$    EXTEND ( / TheRule / )
      COMMON / TheRule / ITRIPLE, NTRIPLE(3,1000)
*
*  MULTIPLICATION TABLE FOR <NVAR> VARIABLES AND <NORDER> ORDER OF
*  DERIVATIVES IN DIFFERENTIAL ALGEBRA.
*  REQUIRES PRIOR EXECUTION OF INTEGER FUNCTION <NTOTAL> TO DETERMINE <N>
*  AND TO SET UP TABLE OF MONOMIAL PAIRS IN <NTRIPLE>. THERE ARE <ITRIPLE>
*  NON-TRIVIAL PAIRS STORED IN EXTENSIBLE COMMON BLOCK <TheRule>.
*
*  C = A * B
*
      C(1) = A(1) * B(1)
      DO 1 I = 2, N
    1   C(I) = A(1) * B(I) + B(1) * A(I)
      DO 2 I = 1, ITRIPLE
        C(NTRIPLE(3,I)) = C(NTRIPLE(3,I))
     $                  + A(NTRIPLE(1,I)) * B(NTRIPLE(2,I))
     $                  + B(NTRIPLE(1,I)) * A(NTRIPLE(2,I))
        IF ( NTRIPLE(1,I) .EQ. NTRIPLE(2,I) ) THEN
          C(NTRIPLE(3,I)) = C(NTRIPLE(3,I))
     $                    - A(NTRIPLE(1,I)) * B(NTRIPLE(1,I))
        END IF
    2 CONTINUE
*
      RETURN
      END
```

ISSN 0067-0367

To identify individual documents in the series we have assigned an AECL- number to each.

Please refer to the AECL- number when requesting additional copies of this document

from

Scientific Document Distribution Office
Atomic Energy of Canada Limited
Chalk River, Ontario, Canada
K0J 1J0

Price: A

ISSN 0067-0367

Pour identifier les rapports individuels faisant partie de cette série nous avons assigné un numéro AECL- à chacun.

Veuillez faire mention du numéro AECL- si vous demandez d'autres exemplaires de ce rapport

au

Service de Distribution des Documents Officiels
Énergie atomique du Canada limitée
Chalk River, Ontario, Canada
K0J 1J0

Prix: A

2009-90