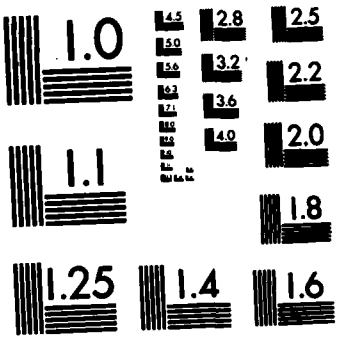
AD-A125 246 DIFFERENTIATION AND GENERATION OF TAYLOR COEFFICIENTS IN PASCAL-SC(U) WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER L B RALL DEC 82 MRC-TSR-2452 1/1

UNCLASSIFIED DAAG29-80-C-0041 F/G 12/1 NL



MRC Technical Summary Report #2452

DIFFERENTIATION AND GENERATION OF TAYLOR COEFFICIENTS IN PASCAL-SC

L. B. Rall

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

December 1982

(Received November 9, 1982)

Approved for public release
Distribution unlimited

Sponsored by

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

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

DIFFERENTIATION AND GENERATION OF TAYLOR COEFFICIENTS IN PASCAL-SC

L. B. RALL

Technical Summary Report #2452
December 1982

ABSTRACT

Evaluation of derivatives and Taylor coefficients of functions defined by computer programs has many applications in scientific computation. The process of automatic differentiation of such functions has as its goal the production of machine code for the evaluation of derivatives and Taylor coefficients. This is in contrast to symbolic differentiation, where the desired output is a more or less pretty formula, and numerical differentiation, which is inaccurate and unstable. Another distinction between automatic and symbolic differentiation is that the latter usually involves considerable computational overhead, while automatic differentiation can be carried out at compile time by a compiler which permits user-defined data types and operators. This report shows how PASCAL-SC, a compiler of this type, can be used to generate the real derivative types GRADIENT, HESSIAN, TAYLOR, and the corresponding interval types IGRADIENT, IHESSIAN, ITAYLOR. Applications of these types to solution of systems of nonlinear equations, sensitivity analysis, constrained and unconstrained optimization, and the solution of initial-value problems for systems of ordinary differential equations are indicated.

AMS (MOS) Subject Classifications: 65G10, 65H10, 65H15, 65K10, 65L05, 65V05, 68C20

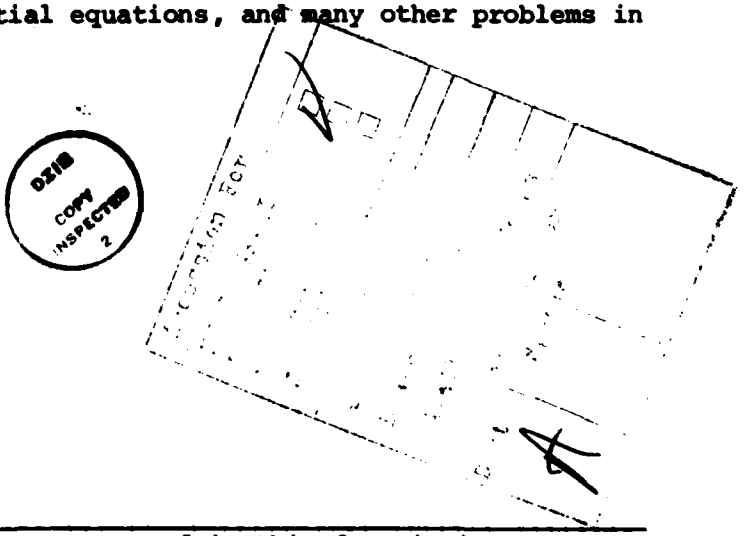
Key Words: Scientific computation, automatic differentiation, generation of Taylor coefficients, gradients, Hessians, systems of equations, sensitivity analysis, constrained and unconstrained optimization, interval analysis, error analysis, PASCAL-SC

Work Unit Number 3 - Numerical Analysis

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

SIGNIFICANCE AND EXPLANATION

Many problems in applied mathematics can be solved by the use of derivatives and Taylor series. While such methods appear to be effective in theory, their application in scientific computation has been limited until recently by the need to code the required derivatives by hand, even though differentiation is a well-understood mechanical process. Thus, the user of such methods has had the choice of resorting to numerical differentiation, an inaccurate and unstable process, or lately to the use of codes for symbolic operations which include differentiation. The latter tend to be large, unwieldy systems, intended to produce formulas and perform a number of symbolic manipulations in addition to differentiation. This report, on the other hand, describes the use of a modern compiler which permits user-defined data types and operators in order to perform automatic differentiation of computed functions in a fast, neat way. Here, the compiler itself, and not some outside system, does the differentiation and generation of Taylor coefficients desired, and produces compact, efficient machine code for their evaluation. The results are immediately applicable to problems including the solution of systems of nonlinear equations, sensitivity analysis, constrained and unconstrained optimization, the solution of initial-value problems for systems of ordinary differential equations, and many other problems in scientific computation.



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

DIFFERENTIATION AND GENERATION OF TAYLOR COEFFICIENTS IN PASCAL-SC

L. B. Rall

1. Automation of evaluation and differentiation of functions. The *value* f(x) of a function of n variables, $x = (x_1,\ldots,x_n)$ is obtained on a computer by the composition $f = f_1 \circ f_2 \circ \ldots \circ f_m$ of a finite number of functions, each usually very simple, such as an arithmetic operation or one of a small number of library functions. Using the fact that the rule for evaluation of each f_i is known, it is possible for a computer program called a compiler to produce machine code for the evaluation of a function f specified in some form similar to ordinary mathematical notation, and including the possibility of different cases depending on x and intermediate results. A key feature of such compilers is the ability to perform *formula translation*, that is, to produce machine code for the evaluation of an *expression* such as

(1.1) `F := (X*Y + SIN(X) + 4.0)*(3*(Y**2) + 6);`

which denotes the mathematical function

(1.2) $$f(x,y) = (xy + \sin x + 4)(3y^2 + 6).$$

The same strategy can be applied for the machine evaluation of derivatives f'(x) or Taylor coefficients of f at x [6], [9], [10], [11], [12], [13]. The chain-rule of calculus applied to f gives

(1.3) $$f'(x) = f_1'(x^{(m-1)})\cdot\ldots\cdot f_{m-1}'(x^{(1)})\cdot f_m'(x),$$

where the ' in general denotes Fréchet differentiation and the · matrix multiplication [11], and

---

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

(1.4) $$x^{(k)} = f^{(k)}(x), \quad f^{(k)} = f_{m-k+1} \circ \ldots \circ f_m, \quad k = 1,\ldots,m-1.$$

Since the rules for calculation of derivatives and Taylor coefficients for the basic arithmetic operations are perfectly well understood, explicit expressions are available for the functions f_i' [13], and thus machine code can be generated easily for the evaluation of derivatives or Taylor coefficients of f at points x for which it is differentiable or analytic, respectively. Most existing compilers, however, do not present this opportunity to the user. The reason is that most were developed only to work with integers and floating-point numbers (i.e., the types INTEGER and REAL) originally, and are quite limited in the kinds of data and operations which can be handled.

PASCAL-SC [3], on the other hand, was developed with the needs of scientific computation in mind, and thus provides complex numbers, intervals, complex intervals, and vectors and matrices over these types, as well as the appropriate operations, in addition to integers, and floating-point numbers, vectors, and matrices [17]. This vast improvement over the ordinary type of compiler is achieved by allowing the introduction of user-defined data types and operators. This facility, as will be explained below, permits automatic differentiation and generation of Taylor coefficients, simply by introduction of the appropriate derivative data types and corresponding operators.

2. Derivative data types. The first derivative of a function f of n variables $x = (x_1,\ldots,x_n)$ is its *gradient vector*

(2.1) $$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n}\right);$$

similarly, the second derivative of f at x is represented by its *Hessian matrix*

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

a symmetric n×n matrix [11]. Furthermore, if f has a convergent Taylor series expansion at $x = x_0$, then

$$f(x) = \sum_{k=0}^{n-1} \frac{1}{k!} f^{(k)}(x_0)(x - x_0)^k + R_n(x_0, x-x_0), \tag{2.3}$$

where, of course, $\frac{1}{0!} f^{(0)}(x_0) = f(x_0)$ [11]. In the scalar case, x is a single real variable, and f(x) is approximated by the *Taylor polynomial*

$$P_{n-1}(x_0, x-x_0) = \sum_{k=0}^{n-1} \frac{1}{k!} f^{(k)}(x_0)(x - x_0)^k, \tag{2.4}$$

which in turn can be represented by the n-vector

$$p_n(x_0, t) = (f_1, f_2, \ldots, f_n) \tag{2.5}$$

of *normalized Taylor coefficients*

$$f_i = \frac{1}{(i-1)!} f^{(i-1)}(x_0) t^{i-1}, \quad t = x - x_0, \tag{2.6}$$

of f. Thus, in the real case, real vectors and matrices are required for the representation of derivatives and Taylor coefficients. Similarly, interval vectors and matrices will be necessary to represent the same quantities for interval-valued functions. These types are all provided in PASCAL-SC [17]. The declaration for vectors in the real case (RVECTOR) runs as follows:

```
CONST DIM =  ;
TYPE DIMTYPE = 1..DIM;
     RVECTOR = ARRAY[DIMTYPE]OF REAL;
```
(2.7)

To declare a real matrix (RMATRIX), one adds

```
TYPE RMATRIX = ARRAY[DIMTYPE]OF RVECTOR;
```
(2.8)

to the above. These declarations will be basic to the following discussion.

2.1. Type GRADIENT. The basic derivative type treats the value f(x) of f at x and its gradient vector $\nabla f(x)$ at the same point as the entity $(f(x), \nabla f(x))$, that is, the pair consisting of the real value of f at x, and its n-dimensional gradient vector of partial derivatives. For this purpose, the RECORD data structure of PASCAL-SC is ideal. To declare this type, (2.7) is followed by

(2.9) TYPE GRADIENT = RECORD F: REAL; DF: RVECTOR END;

so that if V is a variable of type GRADIENT, then V.F will be its value, and V.DF its gradient vector. Thus, if the function v is denoted in the computer program as V, then V.F = v(x) and V.DF = ∇v(x) at the current value x of the independent variables $x_1,\ldots,x_n$. Thus, V.DF[i] = $\partial v(x)/\partial x_i$, i = 1,...,n. The ith independent variable x_i will be denoted by a GRADIENT variable, say XI, such that XI.F = x_i, the current value of x_i, and XI.DF is the ith unit vector e_i = (0,...,0,1,0,...,0) which has its ith component equal to 1, and the others equal to zero. If it is desired to represent a constant c as a GRADIENT variable C, then C.F = c, while C.DF will be the zero vector (0,...,0). The user of type GRADIENT is free to name and order the independent variables arbitrarily.

2.2. Type HESSIAN. This type can be considered to be an extension of type GRADIENT. Here, operations are performed on the triple (f(x),∇f(x),Hf(x)) as the basic datum. Once again, the RECORD structure is appropriate, and the declaration of this type consists of (2.7) followed by (2.8) and

(2.10) TYPE HESSIAN = RECORD F: REAL; DF: RVECTOR; HF: RMATRIX END;

with now V.F = v(x), V.DF = ∇v(x), and V.HF = Hv(x) for a variable V of type HESSIAN corresponding to the function v. Here, V.HF[i , j] = $\partial^2 v(x)/\partial x_i \partial x_j$, for example.

2.3. Type TAYLOR. As indicated above, the quantity to be computed with in this case is the vector (2.5) of normalized Taylor coefficients (2.6). It is important in most applications to be aware of the *scale factor* t = x - x_0, which will be of type REAL in the scalar case. Thus, the RECORD structure will also be used for this derivative type, which is declared by (2.7) followed by

(2.11) TYPE TAYLOR = RECORD T: REAL; TC: RVECTOR END;

with V.T = x - x_0 and V.TC[i] = $\frac{1}{(i-1)!} v^{(i-1)}(x_0)(x - x_0)^{i-1}$ for i = 1,...,n. The value of the Taylor polynomial (2.4) can be obtained very simply and accurately in PASCAL-SC by use of the SUM function:

(2.12) P := SUM(V.TC,Ø);

where P is of type REAL. In (2.12), the sum is rounded to the closest floating-point number; upward or downward rounding can be achieved by replacing the 0 by +1 or -1, respectively [17].

2.4. Interval types IGRADIENT, IHESSIAN, ITAYLOR. Interval arithmetic, including operations for interval vectors and matrices and standard functions, is another convenient feature of PASCAL-SC [3], [17]. By the use of interval computation, inclusions of the real values of expressions such as (1.1) can be obtained [9], [10]. The same applies to values of derivatives and Taylor coefficients, so the derivative types introduced above can also be defined over intervals. The standard declaration of an interval in PASCAL-SC is:

(2.13) TYPE INTERVAL = RECORD INF,SUP: REAL END;

and interval vectors and matrices are declared by

(2.14)
```
TYPE IVECTOR = ARRAY[DIMTYPE]OF INTERVAL;
     IMATRIX = ARRAY[DIMTYPE]OF IVECTOR;
```

corresponding to (2.7)-(2.8). The INTERVAL versions of the REAL derivative types GRADIENT, HESSIAN, and TAYLOR are thus declared by

(2.15)
```
TYPE IGRADIENT = RECORD IF:INTERVAL; IDF:IVECTOR END;
     IHESSIAN  = RECORD IF:INTERVAL; IDF:IVECTOR; IHF:IMATRIX END;
     ITAYLOR   = RECORD IT:INTERVAL; ITC:IVECTOR END;
```

respectively.

3. Derivative operators. In order for expressions to be evaluated correctly when derivative data types appear, the arithmetic operations and standard functions have to be defined in such a way as to incorporate the appropriate rules for differentiation or generation of Taylor coefficients. Furthermore, as in (1.1), provision for variables or constants of type INTEGER or REAL must be made. The following rule

applies:

Variables or expressions of type REAL *or* INTEGER *will be treated as* constants *for the purpose of differentiation.*

In order to simplify the following discussion, generic variables of type INTEGER will be denoted by K, and of type REAL by R, RA, RB. The derivative types introduced in §2 will be indicated by G, H, T, IG, IH, IT, respectively, and the general derivative type by D. Thus, if D = T, type TAYLOR is meant. Generic variables of type D will be denoted by D, DA, DB for D ∈ {G,H,T,IG,IH,IT}. The necessary arithmetic operators are given below; operations between reals and types IG,IH,IT are undefined.

3.1. Addition operators.

(3.1) +D, K+D, D+K, R+D, D+R, DA+DB.

3.2. Subtraction operators.

(3.2) -D, K-D, D-K, R-D, D-R, DA-DB.

3.3. Multiplication operators.

(3.3) K*D, D*K, R*D, D*R, DA*DB.

3.4. Division operators.

(3.4) K/D, D/K, R/D, D/R, DA/DB.

There are thus 22 operators for each real type and 14 for each interval derivative data type. For example, taking D = H, the source code for the operator symbolized by H+R (addition of a REAL to a HESSIAN) is:

```
(3.5)  OPERATOR + (H: HESSIAN;R: REAL) RES: HESSIAN;
          VAR U: HESSIAN;
          BEGIN U.F:=H.F+R;U.DF:=H.DF;U.HF:=H.HF;
                RES:=U
          END;
```

since the addition of a constant alters only the value of a variable, and not its derivatives. Rules for performing the above operations for each derivative type

are given in [13], for example. Source code for each of the real data types G,H,T will be given for the operation DA*DB in the next section. A complete set of source code for arithmetic and power operations and standard functions can be found in [14] for type GRADIENT.

3.5. Power operators. The provision of the power operator ** to evaluate x^y is not standard in PASCAL-SC. In order to implement this operator for derivative data types, it is necessary to define it for some combinations of the basic REAL and INTERVAL types. Thus, one needs

(3.6) R**K, RA**RB,

for type REAL, and, if I,IA,IB denote generic INTERVAL variables,

(3.7) I**K, K**I, IA**IB

for type INTERVAL. In terms of these basic power operations, the derivative operators symbolized by

(3.8) D**K, K**D, D**R, R**D, DA**DB

can be defined. Thus, 5 operators for each of the derivative types are needed, in addition to the 2 operators (3.6) for type REAL, and the 3 operators (3.7) for type INTERVAL, as well as D**K, K**D, DA**DB for $D \in \{IG,IH,IT\}$.

The operator R**K can be implemented by repeated squaring [13], [14], or, better yet, by the accurate algorithm described by Böhm [2]. RA**RB can be calculated using R**K for the integer part of RB, and the ordinary exponential function for the logarithm of the base multiplied by the fractional part of RB. For interval variables X,Y, X^Y is defined by

(3.9) $$X^Y = [\min\{x^y \mid x\in X, y\in Y\}, \max\{x^y \mid x\in X, y\in Y\}],$$

and the operator ** is defined accordingly. Since the inclusion

(3.10) $$X^2 = [\min\{x^2 \mid a \le x \le b\}, \max\{x^2 \mid a \le x \le b\}] \subset \ldots X$$

can be proper for intervals X = [a,b], the standard function ISQR for interval arithmetic [17] is used to compute I**2, and I**K by repeated squaring.

The complete package of arithmetic operations for derivative data types thus consists of 27 operators for each real type, the two real operators (3.6), the three interval operators (3.7), and the 17 arithmetic operators for the interval types.

The priorities of the operators given in this section, from highest to lowest, are: 1°. Unary addition and subtraction, symbolized by +D and -D, respectively; 2°. Multiplication, division, and power: *,/,**; 3°. Binary addition and subtraction ±. Persons familiar with compilers in which ** has higher priority than *,/ should beware, and use parentheses, as in (1.1).

4. Examples of multiplication operators in PASCAL-SC. In order to give explicit examples, the source code for definition of the operator * for the real derivative types GRADIENT, HESSIAN, and TAYLOR will be given in this section, both in component form and compact form using the vector and matrix operators available in PASCAL-SC. Complete source code for the 29 operators required for type GRADIENT is given in [14].

4.1. GA*GB for type GRADIENT. In component form, the source code for this operator is:

```
OPERATOR * (GA,GB: GRADIENT) RES: GRADIENT;
    VAR I: DIMTYPE;U: GRADIENT;
    BEGIN U.F:=GA.F*GB.F;
          FOR I:=1 TO DIM DO U.DF[I]:=GA.F*GB.DF[I]+GB.F*GA.DF[I];
          RES:=U
    END;
```

(4.1)

however, since multiplication of vectors by real numbers and addition of vectors are operations which are available in PASCAL-SC [17], these can be used to produce the more compact source code

```
OPERATOR * (GA,GB: GRADIENT) RES: GRADIENT;
    VAR U: GRADIENT;
    BEGIN U.F:=GA.F*GB.F;
(4.2)           U.DF:=GA.F*GB.DF+GB.F*GA.DF;
                RES:=U
    END;
```

4.2. HA*HB for type HESSIAN. If XI is the ith independent variable of type HESSIAN, then XI.DF will be the ith unit vector, as for type GRADIENT, and XI.HF will be the zero matrix. If a constant C is declared as type HESSIAN, then both the vector part C.DF and the matrix part C.HF of C will be zero. It is, of course, more economical to introduce constants as type INTEGER or REAL. Multiplication of HESSIAN variables HA,HB is accomplished by the operator

```
OPERATOR * (HA,HB: HESSIAN) RES: HESSIAN;
    VAR I,J: DIMTYPE;U: HESSIAN;
    BEGIN
        U.F:=HA.F*HB.F;
        FOR I:=1 TO DIM DO
            U.DF[I]:=HA.F*HB.DF[I]+HB.F*HA.DF[I];
        FOR I:=1 TO DIM DO
(4.3)       FOR J:=I TO DIM DO
            BEGIN
               U.HF[I , J]:=HA.F*HB.HF[I , J]+HA.DF[I]*HB.DF[J]+
                           HB.DF[I]*HA.DF[J]+HB.F*HA.HF[I , J];
               IF I<>J THEN U.HF[J , I]:=U.HF[I , J]
            END;
        RES:=U
    END;
```

in component form, where the well-known formulas for the first and second derivative of products have been used [7], [13]. The calculation of U.DF here is taken directly from (4.1). The first FOR loop could be eliminated by computing U.DF[I] in the second. As before, (4.3) can be simplified by using the vector and matrix operations of PASCAL-SC, augmented by the *outer product* VA**VB of vectors defined

y the operator

```
OPERATOR ** (VA,VB: RVECTOR) RES: RMATRIX;
    VAR I,J: DIMTYPE;U: RMATRIX;
    BEGIN
4.4)    FOR I:= 1 TO DIM DO
          FOR J:=1 TO DIM DO U[I ,J]:=VA[I]*VB[J];
        RES:=U
    END;
```

or example. With this operation, (4.3) becomes

```
OPERATOR * (HA,HB: HESSIAN) RES: HESSIAN;
    VAR U: HESSIAN;
    BEGIN
4.5)      U.F:=HA.F*HB.F;
          U.DF:=HA.F*HB.DF+HB.F*HA.DF;
          U.HF:=HA.F*HB.HF+HA.DF**HB.DF+HB.DF**HA.DF+HB.F*HA.HF;
          RES:=U
    END;
```

hich is more compact but less efficient than (4.3)

4.3. TA*TB for type TAYLOR. Here, the well-know formula for the Taylor oefficients of the product [9], [13] are used. In symbolic form, for U:=TA*TB, ne has ([13], p. 41)

$$4.6) \qquad U.TC[J] = \sum_{I=1}^{J} TA.TC[I]*TB.TC[J-I+1], \quad J = 1,\ldots,DIM.$$

very important advantage of the use of PASCAL-SC for this calculation is that 4.6) can be computed as the scalar product of two vectors; this is done by the tandard function SCALP to the closest floating-point number, or can be rounded up r down if desired. This means, for example, that U.TC[1],...,U.TC[DIM] can each e computed with only one rounding error from the components of TA.TC and TB.TC, nstead of having increasing rounding error as J increases. This helps to ameliorate ne of the possible bugaboos of the use of Taylor series methods in scientific

computation. As can be seen in [13], a number of other formulas for the recursive generation of Taylor coefficients take the form of scalar products, so that one can take advantage of the accuracy of SCALP in their computation. Source code for the multiplication operator * for TAYLOR variables is

```
OPERATOR * (TA,TB: TAYLOR) RES: TAYLOR;
    VAR I,J,K: DIMTYPE;X,Y: RVECTOR;U: TAYLOR;
    BEGIN
      IF TA.T <> TB.T THEN
         BEGIN
           WRITELN('ERROR:  MULTIPLICATION OF TAYLOR VARIABLES WITH
                   UNEQUAL SCALE FACTORS');SVR(Ø)
         END;
      U.T:=TA.T;FOR I:=1 TO DIM DO
         BEGIN X[I]:=Ø;Y[I]:=Ø END;
      FOR I:=1 TO DIM DO
         BEGIN X[I]:=TA.TC[I];
           FOR J:=1 TO I DO
             BEGIN
               K:=I-J+1;Y[J]:=TB.TC[K]
             END;
           U.TC[I]:=SCALP(X,Y,Ø)
         END;
      RES:=U
    END;
```

(4.7)

where an attempt to multiply TAYLOR variables with unequal scale factors results in printing an error message and return of control to the operating system. The initialization of the vectors X,Y could be done by the assignments X:=VRNULL, Y:=VRNULL, from a parameterless function available in PASCAL-SC to produce zero vectors [17].

The present implementation of type TAYLOR is for the scalar case, with a single independent real variable, say x, and expansions are performed at $x = x_0$. In this case, the corresponding independent variable X of type TAYLOR is defined as follows:

(4.8) $\quad X.T = x - x_0,\ X.TC[1] = x_0,\ X.TC[2] = x - x_0,\ X.TC[I] = 0 \text{ for } I > 2.$

Thus, the assignment

(4.9) X.TC[1]:=X.TC[1]+X.TC[2]; or X.TC[1]:=SUM(X.TC,Ø);

will generate a sequence $x_i = x_{i-1} + h$, $h = x - x_0$ of points of expansion of the corresponding TAYLOR variables dependent on X.

Multiplication of the interval derivative types IGRADIENT, IHESSIAN, AND ITAYLOR follows the pattern given above for the corresponding real types.

5. Standard derivative functions. The rules for differentiation of the standard functions ABS, SQR, SQRT, EXP, LN, SIN, COS, ARCTAN are well understood, as is the generation of Taylor coefficients for these functions. Thus, there is no conceptual (or practical) difficulty in providing the standard functions DABS, DSQRT, DEXP, DLN, DSIN, DCOS, DARCTAN for the derivative types D ∈ {G,H,T,IG,IH,IT}, and the function DSQR for the interval derivative types D ∈ {IG,IH,IT}.

Naming functions in this way means that in an expression for the function (1.2), Taylor coefficients are obtained automatically if one declares F,X,Y to be of type TAYLOR, and writes the assignment

(5.1) F := (X*Y + TSIN(X) + 4.0)*(3*(Y**2) + 6);

here, the minor nuisance of writing TSIN instead of SIN is offset by the fact that the former makes it clear that the expression in (5.1) is of type TAYLOR, and that arguments of type TAYLOR are expected.

A complete set of source code for the GRADIENT (D = G) versions of the standard functions listed above is given in [14]. For example, for the gradient cosine,

```
(5.2)  FUNCTION GCOS(G: GRADIENT): GRADIENT;
         VAR M: REAL;U: GRADIENT;
         BEGIN
           M:=-SIN(G.F);U.F:=COS(G.F);U.DF:=M*G.DF;GCOS:=U
         END;
```

in vector form. With the arithmetic operations and the standard functions given, it is very easy for the user to introduce others. If the gradient secant is needed, one

can simply use the code

```
FUNCTION GSEC(G: GRADIENT): GRADIENT;
BEGIN GSEC:=1/GCOS(G) END;
```
(5.3)

for this purpose, and so on.

6. Applications of derivative types in scientific computation. Since the operations of differentiation and series expansion are ubiquitous in mathematical modeling, possible applications of derivative data types in scientific computation would fill a vast catalog, which would also probably never be completed. Here, only a few applications which have been implemented in software in one form or another will be mentioned. In each case, the PASCAL-SC formulation of these types and their corresponding functions and operations leads to a drastic simplification of previous efforts.

6.1. Applications of type GRADIENT and IGRADIENT. A simple application of type GRADIENT is to *sensitivity analysis*. If F is of type GRADIENT, then F.DF[i] = $\partial f/\partial x_i$ is the rate of change of the function f symbolized by F with respect to the ith independent variable x_i. Similarly, F.DF[i]/F.F is the *relative* rate of change of f with respect to x_i. Furthermore, the gradient vector F.DF = $\nabla f(x)$ gives the direction of the fastest increase of f at the current values of the independent variables, a fact which is useful in optimization and other applications.

Given several GRADIENT variables F,G,H, their *Jacobian matrix* J will have rows J[1] = F.DF, J[2] = G.DF, J[3] = H.DF. Knowing the values F.F, G.F, and H.F of these variables as well as their Jacobian matrix makes it very easy to code the solution of systems of equations by Newton's method in terms of GRADIENT variables [7], [11], [14]. Furthermore, the type IGRADIENT can be used to obtain Lipschitz constants for functions of several variables [14].

In interval analysis [9], [10], it is well-known that the mean-value form

$$F(X) = f(x) + F'(X)(X - x) \tag{6.1}$$

can give an accurate interval inclusion of a real function f on a small interval

X containing x. If f is a function of several variables, an interval inclusion F'(X) of the Jacobian matrix f'(x) is needed. This can be obtained automatically if f is coded n terms of its components f_i as variables FI of type IGRADIENT.

By use of PASCAL-SC and the types GRADIENT and IGRADIENT, the program NEWTON [7] for the solution of nonlinear systems of equations can be reduced from over 3,400 lines in FORTRAN to a few dozen.

6.2. Applications of type HESSIAN and IHESSIAN. As one might expect, these types appear to be extremely useful in optimization. In unconstrained optimization, the gradient vector $\nabla f(x)$ of a function $f(x) = f(x_1,\ldots,x_n)$ will vanish at a maximum or minimum value of f, so one wants to solve the system of equations $\nabla f(x) = 0$, the Jacobian matrix of which is $Hf(x)$. If f is coded as a HESSIAN variable F, that is, by an assignment of the form

```
(6.2)          F := F(Xl,...,XN);
```

then the components of the gradient vector are given automatically by F.DF, and the Hessian matrix of f by F.HF, the latter being useful in solution procedures based on Newton's method [14]. Suppose that the optimization problem is *constrained* by conditions $g(x_1,\ldots,x_n) = 0$, $h(x_1,\ldots,x_n) = 0$, where g,h are coded in symbolic form by the assignments

```
(6.3)          G := G(Xl,...XN);  H := H(Xl,...,XN);
```

as variables of type HESSIAN. Then, increasing the dimension of the problem to n+2 and introducing two new independent variables Ll, L2 (the *Lagrange multipliers*) as of type HESSIAN, one makes the assignment

```
(6.4)          W := F + Ll*G + L2*H;
```

and the solution of the unconstrained problem for W is then a solution of the constrained problem for F [14]. Once again, the Hessian matrix W.HF of W is the Jacobian matrix of the system W.DF = 0, which gives a necessary condition for an extreme value of W, and can be used in solution procedures based on Newton's method.

In the program NEWTON [7], interval Hessian matrices were used to obtain Lipschitz constants for Jacobian matrices and ultimately rigorous error bounds for approximate solutions of nonlinear systems of equations. These matrices can be obtained automatically if the type IHESSIAN is used in the computation [14].

6.3. Applications of type TAYLOR and ITAYLOR. Taylor series and the approximation of functions by Taylor polynomials appear in many places in scientific computation. One of the most successful areas of application of the idea of recursive generation of Taylor coefficients is the numerical solution of the initial-value problem for ordinary differential equations and systems of such equations [1], [4], [9], [10]. Type TAYLOR is ideally suited for production of software for this purpose. For example, suppose a Taylor polynomial approximation is sought for the solution of the equation

$$y' = (xy + \sin x + 4)(3y^2 + 6), \quad y(x_0) = y_0, \tag{6.5}$$

say on the interval $x_0 \leq x \leq x_T$. Part of the PASCAL-SC code for this purpose, with X, Y, YPRIME of type TAYLOR, would run as follows for a REAL step-length of H:

```
BEGIN  X.T:=H;Y.T:=H;X.TC:=VRNULL;X.TC[1]:=XØ;X.TC[2]:=H;
  WHILE X.TC[1] <= XT DO
     BEGIN  Y.TC:=VRNULL;Y.TC[1]:=YØ;
       FOR I:=2 TO DIM DO
          BEGIN
           YPRIME:=(X*Y+TSIN(X)+4)*(3*(Y**2)+6); J:=I+1;
           Y.TC[I]:=YPRIME.TC [J]*H/J
          END;
       X.TC[1]:=X.TC[1]+H;YØ:=SUM(Y.TC,Ø);
       WRITELN(X.TC[1],YØ)
     END;
END;
```

(6.6)

The same kind of coding can be expanded for systems of equations.

In the program DIFEQ [4], interval Taylor coefficients and polynomials are

used for rigorous error and step-size control in the computation analogous to (6.6) for the recursive generation of Taylor coefficients of y from those for y'. In the program INTE [5], interval Taylor coefficients are used to obtain guaranteed bounds for the error terms in various rules of numerical integration. In both of these cases, the use of the type ITAYLOR would result in the reduction of programs with thousands of lines in FORTRAN and assembly language to PASCAL-SC programs of a hundred lines or so.

A challenge in the extension of type TAYLOR to the vector case is that derivatives are then multilinear operators [11], so that both storage requirements and operation times increase drastically. However, some of the present and projected "supercomputers" may well be suitable for Taylor series methods to be applied to partial differential equations along the lines indicated here, given compilers with the capabilities of PASCAL-SC which can also exploit any parallelism available in the hardware.

REFERENCES

1. D. Barton, I. M. Willers, and R. V. M. Zahar. Taylor series methods for ordinary differential equations - an evaluation, [16], pp. 369-390 (1971).

2. H. Böhm. Evaluation of arithmetic expressions with maximum accuracy, Proceedings of the 10th IMACS World Congress on System Simulation and Scientific Computation, Vol. 1, pp. 391-393, Montreal, 1982.

3. G. Bohlender, K. Grüner, E. Kaucher, R. Klatte, W. Krämer, U. W. Kulisch, S. M. Rump, Ch. Ullrich, J. Wolff von Gudenberg, and W. L. Miranker. PASCAL-SC: A PASCAL for contemporary scientific computation, Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., 1981.

4. J. A. Braun and R. E. Moore. A program for the solution of differential equations using interval arithmetic (DIFEQ) for the CDC 3600 and 1604, MRC Tech. Summary Rept. No. 901, University of Wisconsin-Madison, 1968.

5. Julia H. Gray and L. B. Rall. INTE: A UNIVAC 1108/1110 program for numerical integration with rigorous error estimation, MRC Tech. Summary Rept. No. 1428, University of Wisconsin-Madison, 1975.

6. G. Kedem. Automatic differentiation of computer programs, ACM Trans. Math. Software 6, 2 (1980), 150-165.

7. D. Kuba and L. B. Rall. A UNIVAC 1108 program for obtaining rigorous error estimates for approximate solutions of systems of equations, MRC Tech. Summary Rept. No. 1168, University of Wisconsin-Madison, 1972.

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

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

10. R. E. Moore. Methods and Applications of Interval Analysis, SIAM Studies in Applied Mathematics 2, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

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

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

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

14. L. B. Rall. Differentiation in PASCAL-SC: Type GRADIENT, MRC Tech. Summary Rept. No. 2400, University of Wisconsin-Madison, 1982.

15. L. B. Rall. Mean value and Taylor forms in interval analysis, SIAM J. Math. Anal. 14, no. 3 (1983) (to appear); preprint: MRC Tech. Summary Rept. No. 2286, University of Wisconsin-Madison, 1981.

16. J. R. Rice (Editor). Mathematical Software, Academic Press, New York, 1971.

17. J. W[illegible] Gudenberg. Gesamte Arithmetik des PASCAL-SC Rechners: Benutzerhandbuch, Institute for Applied Mathematics, University of Karlsruhe, 1981.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 2452	2. GOVT ACCESSION NO.: AD-A125 240	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): DIFFERENTIATION AND GENERATION OF TAYLOR COEFFICIENTS IN PASCAL-SC		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
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office P.O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE: December 1981 13. NUMBER OF PAGES: 18
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)

Scientific computation, automatic differentiation, generation of Taylor coefficients, gradients, Hessians, systems of equations, sensitivity analysis, constrained and unconstrained optimization, interval analysis, error estimation, PASCAL-SC

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

Evaluation of derivatives and Taylor coefficients of functions defined by computer programs has many applications in scientific computation. The process of automatic differentiation of such functions has as its goal the production of machine code for the evaluation of derivatives and Taylor coefficients. This is in contrast to symbolic differentiation, where the desired output is a more or less pretty formula, and numerical differentiation, which is inaccurate and unstable. Another distinction between automatic and symbolic differentiation is that the latter usually involves considerable computational overhead, while

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

automatic differentiation can be carried out at compile time by a compiler which permits user-defined data types and operators. This report shows how PASCAL-SC, a compiler of this type, can be used to generate the real derivative types GRADIENT, HESSIAN, TAYLOR, and the corresponding interval types IGRADIENT, IHESSIAN, ITAYLOR. Applications of these types to solution of systems of nonlinear equations, sensitivity analysis, constrained and unconstrained optimization, and the solution of initial-value problems for systems of ordinary differential equations are indicated.