Comp. & Maths. with Appls. Vol. 12A, No. 11, pp. 1091–1103, 1986
Printed in Great Britain.
0886-9553/86 $3.00 + 0.00

AUTOMATIC DIFFERENTIATION OF FUNCTIONS OF DERIVATIVES

R. KALABA and L. TESFATSION†
Modeling Research Group, Department of Economics, University of Southern California, Los Angeles, CA 90089-0152, U.S.A.

(*Received November 1985*)

Abstract—In a recent paper an algorithm FEED was introduced for the systematic exact evaluation of higher-order partial derivatives of functions of many variables. The present paper demonstrates that FEED can be applied to a much broader class of functions than envisioned previously. Specifically, FEED can be used to evaluate the higher-order partial derivatives of functions which are defined in terms of the *derivatives* of other functions, a task required in many applications.

1. INTRODUCTION

In a recent paper [1] a "table algorithm" is introduced for the automatic evaluation of higher-order partial derivatives. Building on a key idea of Wengert [2], the evaluation of a complicated kth-order differentiable function F at a specified domain point x is decomposed into a sequence of simple evaluations of special functions of one or two variables. The evaluation of each special function is accomplished by calling a calculus subroutine which automatically computes and returns the distinct partial derivatives of the special function through order k, along with the special function value. The final stage in the sequence of special function evaluations yields the distinct partial derivatives of F at x through order k, along with the function value $F(x)$.

The special functions considered in Ref. [1] for function decomposition are the two-variable functions

$$u + v, \quad u - v, \quad uv, \quad u/v \quad \text{and} \quad u^r, \tag{1}$$

and arbitrary one-variable continuously differentiable functions, such as

$$\sin(u), \quad \cos(u), \quad \exp(u), \quad \log(u) \quad \text{and} \quad au^b + c \tag{2}$$

for arbitrary constants a, b and c. For example, as detailed in Section 2, evaluation of the function

$$z = F(x, y) = x + \log(xy) \tag{3}$$

and its distinct higher-order partial derivatives at a given domain point (x, y) is accomplished by calling, in order, calculus subroutines for the special functions

$$a = x,$$

$$b = y,$$

$$c = ab,$$

$$d = \log(c)$$

and

$$z = a + d. \tag{4}$$

†To whom all correspondence should be addressed.

The calculus subroutines called for the special functions listed in (4) return well-ordered† arrays of function and derivative evaluations, using previously obtained well-ordered arrays as inputs. For example, the calculus subroutine called for the special function $c = ab$ listed in (4) returns the well-ordered array $C = (c, c_x, c_y, c_{xx}, c_{xy}, \ldots)$ for the value and distinct partial derivatives of c, using the previously calculated well-ordered arrays $A = (a, a_x, a_y, a_{xx}, a_{xy}, \ldots)$ and $B = (b, b_x, b_y, b_{xx}, b_{xy}, \ldots)$ as inputs. Thus, $c = ab$, $c_x = a_x b + ab_x$, $c_y = a_y b + ab_y$ and so forth. By construction, the well-ordered array $Z = (z, z_x, z_y, z_{xx}, z_{xy}, \ldots)$ obtained for the final special function $z = a + d$ listed in (4) yields the desired evaluations for the value and distinct partial derivatives of the basic function of interest, $z = x + \log(xy)$.

The present paper demonstrates that the table algorithm, or FEED ("fast efficient evaluation of derivatives") as we now refer to it, can be applied to a much broader class of functions than envisioned previously [1]. Specifically, at each stage in the sequential evaluation of a function via FEED, certain special function *derivatives* are evaluated which in turn can be used as special functions in subsequent stages. Thus, FEED can be used to differentiate functions which are defined in terms of the partial derivatives of other functions.

For example, as explained more fully in Section 3, evaluation of the function

$$u = G(x, y) = x + \frac{\partial \log(xy)}{\partial x} \tag{5}$$

and its distinct higher-order partial derivatives at a given domain point (x, y) can be accomplished by calling, in order, calculus subroutines for the special functions

$$a = x,$$

$$b = y$$

$$c = ab,$$

$$d = \log(c),$$

$$e = d_x$$

and

$$u = a + e. \tag{6}$$

---

†An array Z containing the value and distinct partial derivatives through order k of a kth-order continuously differentiable real-valued function $z = F(x_1, \ldots, x_n)$ at a given domain point $x = (x_1, \ldots, x_n)$ will be referred to as *well-ordered (for F through order k)* if Z has the following form:

$$Z = (F, A_1, A_2, \ldots, A_k),$$

where A_i is a row vector consisting of all the distinct partial derivatives of F at x of order i which satisfies the following two properties:

(a) If $F_{IJ\ldots L}$ appears in A_i, then $I \leqslant J \leqslant \ldots \leqslant L$;
(b) If $F_{IJ\ldots ML}$ and $F_{I'J'\ldots M'L'}$ both appear in A_i, then the former term precedes the latter term if and only if either $I < I'$, or $I = I'$ and $J < J'$, or ..., or $I = I'$ and $J = J'$ and ... and $M = M'$ and $L < L'$.

For example, if $k = 3$ and $n = 4$, Z is well-ordered if and only if $Z = (F, A_1, A_2, A_3)$, with

$$A_1 = (F_1, F_2, F_3, F_4),$$

$$A_2 = (F_{11}, F_{12}, F_{13}, F_{14}, F_{22}, F_{23}, F_{24}, F_{33}, F_{34}, F_{44})$$

and

$$A_3 = (F_{111}, F_{112}, F_{113}, F_{114}, F_{122}, F_{123}, F_{124}, F_{133}, F_{134}, F_{144},$$
$$F_{222}, F_{223}, F_{224}, F_{233}, F_{234}, F_{244}, F_{333}, F_{334}, F_{344}, F_{444}).$$

As is well-known, given any kth-order continuously differentiable function $F: R^n \to R$, and any integer m such that $1 \leqslant m \leqslant k$, the number of distinct partial derivatives of F of order m at any domain point x is given by

$$\binom{n+m-1}{m} = \frac{[n+m-1]!}{m![n-1]!}.$$

The calculus subroutine called for the special function $e = d_x$ appearing in (6) is a selection routine which returns the value and distinct higher-order partial derivatives of $e = d_x$ in a well-ordered array $\mathrm{E} = (d_x, d_{xx}, d_{xy}, \ldots)$, using, as input, the well-ordered array $\mathrm{D} = (d, d_x, d_y, d_{xx}, d_{xy}, \ldots)$ obtained for d in the preceding stage.

The explicit extension of FEED to permit the evaluation of functions of derivatives is potentially useful, since the differentiation of functions of derivatives occurs naturally in a wide variety of applications. For example, given certain regularity conditions, the Euler–Lagrange first-order necessary conditions for the simplest calculus of variations problem

$$\min_x \int_0^T I[\dot{x}(t), x(t)]\,\mathrm{d}t, \quad \text{subject to } x(0) = x_0 \quad \text{and} \quad x(T) = x_T, \tag{7}$$

take the form

$$\ddot{x} = G(\dot{x}, x) = \frac{I_x(\dot{x}, x) - I_{\dot{x}x}(\dot{x}, x)\dot{x}}{I_{\dot{x}\dot{x}}(\dot{x}, x)} \tag{8}$$

for each time $t, 0 \leqslant t \leqslant T$. Thus, $\ddot{x}$ is a function of the partial derivatives of I. Solution of problem (7) by means of quasilinearization or Newton's method requires the evaluation of $\ddot{x}$ and the first-order partial derivatives of $\ddot{x}$ with respect to both $\dot{x}$ and x.

As another example, suppose one wants to obtain the power series expansion of a function $y(x)$ about a point x^0, where $y(x)$ satisfies the nonlinear differential equation

$$y'(x) = G[x, y(x)], \tag{9a}$$

subject to

$$y(x^0) = y^0. \tag{9b}$$

The higher-order derivatives of y' required by the power series expansion are then given by

$$y'' = G_x + G_y G; \tag{10a}$$

and

$$y''' = [G_x + G_y G]_x + [G_x + G_y G]_y G \tag{10b}$$

and so forth. Thus, the higher-order derivatives of y' are functions of the higher-order partial derivatives of G. (See Section 4 for further discussion of this example.)

The basic FEED algorithm [1] is illustrated in Section 2 for the function (3). The extension of FEED to functions of derivatives is illustrated in Section 3 for the function (5). Section 4 discusses the extended FEED algorithm for general functions of higher-order derivatives. Concluding comments are given in Section 5.

As a historical note, the need for an automatic method for evaluating partial derivatives quickly became apparent in the process of developing computational methods for nonlocal comparative static analysis [3]. Use was first made of a method developed by Wengert [2] for sequentially evaluating higher-order partial derivatives. Wengert's key idea was to decompose the evaluation of complicated functions of many variables into a sequence of simpler evaluations of special functions of one or two variables. Total differentials of the special functions could be automatically evaluated along with the special function values. Partial derivatives could then be recovered from the total differentials by solving certain associated sets of linear algebraic equations.

Although programs were successfully written for implementing Wengert's method for first- and second-order partial derivatives, two principal difficulties arose in attempting to extend Wengert's method to higher-order partial derivatives. First, Wengert's method requires the repeated evaluation of certain identical functional forms as each individual partial derivative is separately recovered from a total differential, resulting in significant computational inefficiency. Second, Wengert's method requires the formation and solution of a distinct set of linear algebraic equations for each successively higher-order partial

differentiation, and it does not seem possible to provide a systematic rule for how this is to be done.

The table (FEED) algorithm developed in Ref. [1] for the systematic exact evaluation of higher-order partial derivatives overcomes both of these difficulties. Wengert's key idea of sequential function evaluation is retained, but total differentials and linear algebraic equations play no role. An additional advantage of the FEED algorithm is that memory and arithmetic requirements can be determined prior to any calculations.

The FEED algorithm is conceptually and computationally straightforward, and it is difficult to believe that previous researchers have not already discovered it. However, no presentations of such an algorithm have been found by us in the previous journal literature, or conveyed to us by readers of Ref. [1]. Even Wengert's method is rarely cited. (See, for example, Ref. [4].)

FEED has now been successfully tested in a variety of applications, including nonlinear least squares, optimal control, system identification, nonlinear two-point boundary value problems and the numerical solution of nonlinear integral equations. Discussion of these applications can be found in Ref. [5], and especially in Ref. [6].

Recently [7] an important improvement in the implementation of FEED has been proposed. Rather than explicitly listing a sequence of statements to evaluate a function and its partial derivatives, as illustrated above in equations (4) and (6), the user simply writes an assignment statement which closely resembles the form of the function to be evaluated. For example, to obtain the value and partial derivatives of the function $z = x + \log(xy)$, the user writes an assignment statement of the form

$$\text{K = IADD(IX, ILOGG(IMULT(IX,IY)))}. \tag{11}$$

The FORTRAN compiler then evaluates expression (11), automatically calling the needed calculus subroutines in correct sequence. The integer K is a pointer which indicates where in the memory the function and derivative values (z, z_x, z_y, z_{xx}) are stored.

2. THE BASIC FEED ALGORITHM: A SIMPLE ILLUSTRATION

Consider the function $F: R^2_{++} \to R$, defined by

$$z = F(x, y) = x + \log(xy). \tag{12}$$

Suppose one wishes to evaluate the function value z, the first-order partial derivatives z_x and z_y and the second-order partial derivative z_{xx} at a given domain point (x, y). Consider Table 1.

The first column of Table 1 sequentially evaluates the function value $z = x + \log(xy)$ at the given domain point (x, y). The second, third and fourth entries in each row of Table 1 give the indicated derivative evaluations of the first entry in the row, using only algebraic operations. The first two rows initialize the algorithm, one row being required for each function variable. The only input required for the first two rows is the domain point (x, y). Each subsequent row outputs a well-ordered array of the form (p, p_x, p_y, p_{xx}), using the well-ordered arrays obtained from previous row calculations as inputs. The final row yields the desired evaluations (z, z_x, z_y, z_{xx}). These evaluations are exact up to round-off error.

Table 1. Sequential evaluation of $z = x + \log(xy)$, z_x, z_y and z_{xx}

Special function	$\frac{\partial}{\partial x}$	$\frac{\partial}{\partial y}$	$\frac{\partial^2}{\partial x^2}$
$a = x$	$a_x = 1$	$a_y = 0$	$a_{xx} = 0$
$b = y$	$b_x = 0$	$b_y = 1$	$b_{xx} = 0$
$c = ab$	$c_x = a_x b + ab_x$	$c_y = a_y b + ab_y$	$c_{xx} = a_{xx} b + 2a_x b_x + ab_{xx}$
$d = \log(c)$	$d_x = c^{-1} c_x$	$d_y = c^{-1} c_y$	$d_{xx} = -c^{-2} c_x^2 + c^{-1} c_{xx}$
$z = a + d$	$z_x = a_x + d_x$	$z_y = a_y + d_y$	$z_{xx} = a_{xx} + d_{xx}$

It will now be shown how the rows of Table 1 can be sequentially evaluated by means of FORTRAN calculus subroutines, given any specified domain point (x, y).

The first two rows of Table 1 can be evaluated by calling calculus subroutines LIN1 and LIN2, respectively, defined as follows:

```
SUBROUTINE LIN1(X,A)
DIMENSION A(4)
A(1) = X
A(2) = 1.0
A(3) = 0.0
A(4) = 0.0
RETURN
END
```

```
SUBROUTINE LIN2(Y,B)
DIMENSION B(4)
B(1) = Y
B(2) = 0.0
B(3) = 1.0
B(4) = 0.0
RETURN
END.
```
(13)

The subroutine LIN1 uses standard calculus formulas to obtain the value and first three partial derivatives $A = (a, a_x, a_y, a_{xx}) = (X, 1, 0, 0)$ of the function a of x and y defined by $a(x, y) = x$, given any particular value X for x. Thus, the output array A evaluates the first row of Table 1 at X. Similarly, subroutine LIN2 obtains the value and first three partial derivatives $B = (b, b_x, b_y, b_{xx}) = (Y, 0, 1, 0)$ of the function b of x and y defined by $b(x, y) = y$, given any particular value Y for y. Thus, the output array B evaluates the second row of Table 1 at Y.

Row three of Table 1 can be evaluated by calling the following calculus subroutine for multiplication:

```
SUBROUTINE MULT(A,B,C)
DIMENSION A(4), B(4), C(4)
C(1) = A(1)*B(1)
C(2) = A(2)*B(1) + A(1)*B(2)
C(3) = A(3)*B(1) + A(1)*B(3)
C(4) = A(4)*B(1) + 2.0*A(2)*B(2) + A(1)*B(4)
RETURN
END.
```
(14)

The subroutine MULT uses standard calculus formulas to obtain the value and first three partial derivatives $C = (c, c_x, c_y, c_{xx})$ of the function $c = ab$, where a and b are any two real-valued, twice-differentiable functions of x and y, and the input arrays $A = (a, a_x, a_y, a_{xx})$ and $B = (b, b_x, b_y, b_{xx})$ are well-ordered arrays for a and b. For the case at hand, the input arrays A and B are the outputs of rows 1 and 2 of Table 1.

Row 4 of Table 1 can be evaluated by calling the following calculus subroutine for the log function:

```
SUBROUTINE LOGG(C,D)
DIMENSION C(4), D(4)
D(1) = ALOG(C(1))
D(2) = C(2)/C(1)
D(3) = C(3)/C(1)
D(4) = -D(2)**2 + C(4)/C(1)
RETURN
END.
```
(15)

The subroutine LOGG uses standard calculus formulas to obtain the value and first three partial derivatives $D = (d, d_x, d_y, d_{xx})$ of the function $d = \log(c)$, where c is any real-valued, positive, twice-differentiable function of x and y, and the input array $C = (c, c_x, c_y, c_{xx})$ is a well ordered array for c. For the case at hand, the input array C is the output of row 3 of Table 1. Note that the calculus subroutine LOGG calls the FORTRAN library function subroutine ALOG for the log function.

Finally, row 5 of Table 1 can be evaluated by calling the following calculus subroutine for addition:

```
      SUBROUTINE ADD(A,D,Z)
      DIMENSION A(4), D(4), Z(4)
      DO 1 I=1,4
1       Z(I)=A(I)+D(I)
      RETURN
      END.
```
(16)

The subroutine ADD uses standard calculus formulas to obtain the value and first three partial derivatives $Z = (z, z_x, z_y, z_{xx})$ of the function $z = a + d$, where a and d are any two real-valued, twice-differentiable functions of x and y, and the input arrays $A = (a, a_x, a_y, a_{xx})$ and $D = (d, d_x, d_y, d_{xx})$ are well-ordered arrays for a and d. For the case at hand, the input arrays A and D are the outputs of rows 1 and 4 of Table 1, respectively.

The complete sequential evaluation of the rows of Table 1 is accomplished by means of the following master subroutine:

```
SUBROUTINE FUN(X,Y,Z)
DIMENSION A(4), B(4), C(4), D(4), Z(4)
CALL LIN1(X,A)
CALL LIN2(Y,B)
CALL MULT(A,B,C)
CALL LOGG(C,D)
CALL ADD(A,D,Z)
RETURN
END.
```
(17)

Given any specified domain point $(x, y) = (X,Y)$, the subroutine FUN obtains the value and first three partial derivatives (z, z_x, z_y, z_{xx}) of the function $z = x + \log(xy)$ at (X,Y) by means of the indicated sequence of calls to the calculus subroutines (13)–(16). These evaluations are returned in the array Z.

In principle, a FUN subroutine can be constructed to calculate the value and partial derivatives through order k, $k \geq 1$, of any real-valued multivariable function F which can be sequentially evaluated by means of the special two-variable functions (1) and arbitrary, one-variable, kth-order continuously differentiable functions such as (2). Systematic rules for constructing kth-order calculus subroutines for the special functions (1) and (2) are presented in Ref. [1].

3. THE EXTENDED FEED ALGORITHM: A SIMPLE ILLUSTRATION

Consider the function $G: R^2_{++} \to R$, defined by

$$u = G(x, y) = x + \frac{\partial \log(xy)}{\partial x}. \tag{18}$$

In contrast to the function $F(x, y) = x + \log(xy)$ discussed in Section 2, the definition (18) for the function G is expressed in terms of the *partial derivative* of a second function of x and y.

Suppose one wishes to evaluate the function value u and the first-order partial derivative u_x at a given domain point (x, y). Consider Table 2.

As in Table 1, the first column of Table 2 sequentially evaluates the function value $u = x + \partial \log(xy)/\partial x$ at the given domain point (x, y). The second through fourth entries in each row of Table 2 give the indicated derivative evaluations of the first entry in the row. The final row yields the desired evaluations for u and u_x. These evaluations are exact up to round-off error.

Table 2 differs from Table 1 by the appearance in row 5 of an additional type of special function, $e = d_x$, together with its first-order partial derivative $e_x = d_{xx}$. Note, however, that values for the partial derivatives (d_x, d_{xx}) of d were obtained in the fourth row; (e, e_x) is simply a relabeling of these values which prepares them for use in later row calculations. In order to obtain additional partial derivatives of e, one need only expand the dimension of rows 1–4, so that additional partial derivatives of d are evaluated in row 4 as required.

Table 2. Sequential evaluation of $u = x + \partial \log(xy)/\partial x$ and u_x

Special function	$\frac{\partial}{\partial x}$	$\frac{\partial}{\partial y}$	$\frac{\partial^2}{\partial x^2}$
$a = x$	$a_x = 1$	$a_y = 0$	$a_{xx} = 0$
$b = y$	$b_x = 0$	$b_y = 1$	$b_{xx} = 0$
$c = ab$	$c_x = a_x b + a b_x$	$c_y = a_y b + a b_y$	$c_{xx} = a_{xx} b + 2 a_x b_x + a b_{xx}$
$d = \log(c)$	$d_x = c^{-1} c_x$	$d_y = c^{-1} c_y$	$d_{xx} = -c^{-2} c_x^2 + c^{-1} c_{xx}$
$e = d_x$	$e_x = d_{xx}$		
$u = a + e$	$u_x = a_x + e_x$		

The rows of Table 2 can be evaluated by means of calculus subroutines sequentially called by the following master subroutine:

```
SUBROUTINE FUN(X,Y,U)
DIMENSION A(4), B(4), C(4), D(4), E(4), U(4)
CALL LIN1(X,A)
CALL LIN2(Y,B)
CALL MULT(A,B,C)
CALL LOGG(C,D)
CALL DX(D,E)
CALL ADD(A,E,U)
RETURN
END.
```
(19)

The calculus subroutines LIN1, LIN2, MULT, LOGG and ADD are defined by (13)–(16), as in Section 2. However, an additional subroutine DX(D,E) appears in the list of call statements in (19), corresponding to the additional special function $e = d_x$ in Table 2. The subroutine DX(D,E) creates a new left-column special function e from the partial derivative d_x of the previously evaluated special function d, and returns the value and available partial derivatives of e in a well-ordered array.

More precisely, DX(D,E) is a selection subroutine which forms and returns the array $E = (d_x, d_{xx}, Q, Q)$, using the array $D = (d, d_x, d_y, d_{xx})$ as input, where Q is some preselected arbitrarily large real number in common signifying no usable calculation. [If any element of an array is used which takes on the value Q, a warning statement should be printed out.] The FORTRAN statements defining DX(D,E) are as follows:

```
SUBROUTINE DX(D,E)
DIMENSION D(4), E(4)
KK=1
II=1
```
(20a)

```
      DO 1 I=1,4                                          (20b)
    1   E(I)=Q
      DO 2 I=1,2                                          (20c)
        KK=KK+1
        IF (1.EQ.I) GO TO 100
        GO TO 2
  100   E(II)=D(KK)
        II=II+1
    2 CONTINUE
      DO 4 I=1,2                                          (20d)
        DO 3 J=I,2
          KK=KK+1
          IF (1.EQ.I.OR.1.EQ.J) GO TO 200
          GO TO 3
  200     E(II)=D(KK)
          II=II+1
    3   CONTINUE
    4 CONTINUE
      RETURN
      END.
```

Block (20a) initializes the integers KK and II denoting current position in the input array D and output array E, respectively. Block (20b) assigns the initial value Q to each component of E. Block (20c) searches the singly-subscripted elements of the input array $D = (d, d_x, d_y, d_{xx})$ for d_x, and assigns this value to the first component of E. Block (20d) searches the doubly-subscripted elements of the input array D for d_{xx}, and assigns this value to the second component of E.

4. THE EXTENDED FEED ALGORITHM FOR GENERAL FUNCTIONS OF DERIVATIVES

The distinct higher-order partial derivatives of general kth-order continuously differentiable functions $H: R^n \to R$ defined in terms of the higher-order partial derivatives of other real-valued functions on R^n can be evaluated by making repeated calls to a calculus subroutine DERK, briefly described as follows. Given a well-ordered array U containing the value and distinct partial derivatives through order k of a function $u(x_1, \ldots, x_n)$, and given a coordinate subscript L, $L = 1, \ldots, n$, subroutine DERK(U,L,V) selects from U the value and partial derivatives through order $k-1$ of the partial derivative of u with respect to x_L, and returns these evaluations in a well-ordered array V.

For concreteness, consider the function $H: R^2_{++} \to R$ defined by

$$w = H(x, y) = x + \frac{\partial^2 \log (xy)}{\partial x \partial y}. \tag{21}$$

Evaluation of the function (21) and its distinct partial derivatives through any order k at any given domain point (x, y) can be accomplished by calling, in order, calculus

subroutines for the special functions

$$a = x,$$

$$b = y,$$

$$c = ab,$$

$$d = \log(c)$$

$$e = d_x,$$

$$f = e_y$$

and

$$w = a + f. \tag{22}$$

Suppose one wishes to obtain the function value w and the first-order partial derivatives w_x and w_y at (x, y).

The new feature in equations (22) is the presence of *two* consecutive first-order differentiations, $e = d_x$ and $f = e_y$. These differentiations are implemented by consecutive calls to the calculus subroutine DERK(U,L,V). Each call to DERK reduces by one the order of derivatives which can subsequently be obtained for w. Thus, calculus subroutines for the special functions preceding the two consecutive first-order differentiations in equations (22) must obtain evaluations for the distinct partial derivatives of these functions through order 3 if evaluations for the partial derivatives of w through order 1 are to be obtained in the final step.

The master subroutine described below in (23) obtains the desired evaluations (w, w_x, w_y) at (x, y). The dimension M in (23) denotes any actual integer which is greater than 9, the number of distinct partial derivatives through order 3 of a real-valued function of two variables:

```
SUBROUTINE FUN(X,Y,W)
DIMENSION A(M),B(M),C(M),D(M),E(M),F(M),W(M)
CALL LIN1(X,A)
CALL LIN2(Y,B)
CALL MULT(A,B,C)
CALL LOGG(C,D)
CALL DER3(D,1,E)
CALL DER2(E,2,F)
CALL ADD(A,F,W)
RETURN
END.
```
(23)

The subroutines LIN1, LIN2, MULT, LOGG and ADD in (23) are generalizations of the Section 2 calculus subroutines (13)–(16) to dimension M. For example, setting M = 10, subroutine ADD might take the following form:

```
      SUBROUTINE ADD(A,F,W)
      DIMENSION A(10),F(10),W(10)
      DO 1 I=1,10
1       W(I)=Q
      DO 2 I=1,3
2       W(I)=A(I)+F(I)
      RETURN
      END.
```
(24)

Subroutine ADD(A,F,W) calculates the value and first-order partial derivatives (w, w_x, w_y) of the function $w = a + f$, where a and f are any two real-valued, continuously differentiable functions of x and y, and the input arrays $A = (a, a_x, a_y, \ldots)$ and $F = (f, f_x, f_y, \ldots)$ are well-ordered arrays for a and f through order 1. These evaluations for w are returned in a well-ordered array $W = (w, w_x, w_y, Q, \ldots, Q)$. As in Section 3, Q is some preselected arbitrarily large real number in common signifying no usable calculation. The appearance of Q in any calculation should trigger the printout of a warning statement. Discussion of the generally dimensioned calculus subroutines LIN1, LIN2, MULT and LOGG can be found in Ref. [1].

The subroutine DERK is called twice in (23). The call to DER3(D,1,E) selects from the well-ordered M-dimensional input array

$$D = (d, d_x, d_y, d_{xx}, d_{xy}, \ldots, d_{yyy}, Q, \ldots, Q) \tag{25}$$

for d the value and partial derivatives of $e = d_x$, the derivative of d with respect to the first variable x, and returns them in a well-ordered M-dimensional array

$$E = (d_x, d_{xx}, d_{xy}, d_{xxx}, d_{xxy}, d_{xyy}, Q, \ldots, Q) \equiv (e, e_x, e_y, e_{xx}, e_{xy}, e_{yy}, Q, \ldots, Q) \tag{26}$$

for e. The call to DER2(E,2,F) selects from the input array E for e the value and partial derivatives of $f = e_y$, the derivative of e with respect to the second variable y, and returns them in a well-ordered M-dimensional array

$$F = (e_y, e_{xy}, e_{yy}, Q, \ldots, Q) \equiv (f, f_x, f_y, Q, \ldots, Q) \tag{27}$$

for f. Note, in the interpretation of F, that use has been made of the equivalence between the mixed partials e_{xy} and e_{yx} whose indices differ only by a permutation. This equivalence follows from the second-order continuous differentiability of the special function e.

The FORTRAN statements which define the general calculus subroutine DER2(U,L,V) are given below in (28). The input array U is interpreted to be a well-ordered array through order 2 for some real-valued, twice continuously differentiable function u of n variables. The number of variables NVAR $\equiv n$ is assumed to be in common. [For the illustrative example (21), NVAR equals 2.] The integer L signifies that the partial derivative of u with respect to the Lth variable is under consideration. [For the illustrative example (21), L = 1 signifies that the partial derivative with respect to the first variable x is under consideration, and L = 2 signifies that the partial derivative with respect to the second variable y is under consideration.] The dimension M in (28) denotes any actual integer which is greater than the number of distinct partial derivatives of u through order 2:

```
      SUBROUTINE DER2(U,L,V)
      DIMENSION U(M),V(M)
      KK=1                                                    (28a)
      II=1
      DO 1 I=1,M                                              (28b)
    1   V(I)=Q
      DO 2 I=1,NVAR                                           (28c)
        KK=KK+1
        IF (L.EQ.I) GO TO 100
        GO TO 2
```

```
100     V(II) = U(KK)
        II = II + 1
  2     CONTINUE
      DO 4 I = 1, NVAR
        DO 3 J = I,NVAR
          KK = KK + 1
          IF (L.EQ.I.OR.L.EQ.J) GO TO 200
          GO TO 3
200       V(II) = U(KK)
          II = II + 1
  3     CONTINUE
  4   CONTINUE
      RETURN
      END.
```
(28d)

The blocks in (28) are defined analogously to the blocks in (20). [Indeed, DX(D,E) is equivalent to DER2(D,1,E).] Thus, block (28a) initializes the current positions KK and II in the input and output arrays U and V, respectively. Block (28b) assigns the initial value Q to each component of V. Block (28c) searches the singly-subscripted elements of the input array U from left to right for u_L, and assigns this value to the first component of V. Block (28d) searches the doubly-subscripted elements of the input array U from left to right for the first-order partial derivatives of u_L, and assigns each value as found to the next open (i.e. Q-valued) component of the array V.

The question arises whether the array V returned by DER2(U,L,V) is necessarily well-ordered through order 1 for $u_L(x_1, \ldots, x_n)$, given that the input array U is well-ordered through order 2 for $u(x_1, \ldots, x_n)$. The answer is yes, as the following arguments establish.

By assumption, U contains all of the distinct partial derivatives of u through order 2; and, by construction, DER2(U,L,V) sequentially transfers from U to V every singly-subscripted and doubly-subscripted element of U which contains at least one L among its subscripts. Thus, the first component of V is u_L, and the remaining components of V must contain all of the first-order partial derivatives of u_L. Moreover, the subscripts of each of the first-order partial derivatives of u_L must be in ascending order from left to right, since they come from the well-ordered array U.

Suppose the first-order partial derivatives for u_L are not well-ordered for u_L in V. Then there must exist subscripts $I < J$ such that u_{LJ} (or u_{JL}) precedes u_{LI} (or u_{IL}) in the row vector V. For concreteness, suppose $L < I < J$, so that u_{LJ} precedes u_{LI} in V. It then follows, by construction, that u_{LJ} precedes u_{LI} in U; for DER2 *sequentially* transfers elements of U to V from left to right. However, u_{LJ} preceding u_{LI} in U contradicts the initial assumption that U is well-ordered for u through order 2. Similar arguments hold for $I < L < J$ and for $I < J < L$.

Thus, the array V returned by DER2(U,L,V) is necessarily well-ordered through order 1 for u_L if U is well-ordered through order 2 for u.

The calculus subroutine DER3(U,L,V) is a straightforward generalization of the calculus subroutine DER2(U,L,V). Given a well-ordered array U containing the value and distinct partial derivatives through order 3 of a function $u(x_1, \ldots, x_n)$, and given any coordinate subscript L, $L = 1, \ldots, n$, subroutine DER3(U,L,V) selects from U the value and partial derivatives through order 2 of the partial derivative of u with respect to x_L, and returns these evaluations in a well-ordered array V. Thus, in addition to the blocks (28a)–(28d) appearing in subroutine DER2, subroutine DER3 requires the following block of nested DO loops:

```
      DO 7 I=1,NVAR
       DO 6 J=I,NVAR
        DO 5 K=J,NVAR
          KK=KK+1
          IF (L.EQ.I.OR.L.EQ.J.OR.L.EQ.K) GO TO 300
          GO TO 5
300       V(II)=U(KK)
          II=II+1
5       CONTINUE
6     CONTINUE
7   CONTINUE.                                          (29)
```

Block (29) searches the *thrice*-subscripted elements of the input array U from left to right for the *second*-order partial derivatives of u_L, and assigns each value as found to the next open component of V. It is easily established, using arguments similar to those used for DER2(U,L,V), that the array V returned by DER3(U,L,V) is necessarily well-ordered through order 2 for u_L, given that U is well-ordered through order 3 for u.

The procedure for constructing a FORTRAN program for the general calculus subroutine DERK(U,L,V) should now be apparent. Each successively higher integer value k requires the incorporation of one additional block of nested DO loops in the basic program (28) in order to search from left to right through the *k-subscripted* elements of U for elements whose subscripts contain at least one L, i.e. for the partial derivatives of u_L through order $k-1$. The output array V is well-ordered through order $k-1$ for u_L, given that U is well-ordered through order k for u.

As a final illustration of the use of the calculus subroutine DERK, consider the problem of obtaining the power series expansion of a function $y(x)$ about a point x^0, where $y' = x + \log(xy)$, $y(x^0) = y^0$, and x^0 and y^0 are both positive scalars. [See equations (9) and (10) in Section 1.] For concreteness, suppose evaluations are needed for the first three coefficients $y'(x^0)$, $y''(x^0)$ and $y'''(x^0)$. These evaluations can be obtained by calling, in order, appropriately dimensioned calculus subroutines for the special functions:

$$\begin{aligned} a &= x^0, \\ b &= y^0, \\ c &= ab, \\ d &= \log(c), \\ e &= a + d; \end{aligned} \tag{30a}$$

$$\begin{aligned} f &= e_x, \\ g &= e_y, \\ h &= ge, \\ i &= f + h; \end{aligned} \tag{30b}$$

$$\begin{aligned} j &= i_x, \\ k &= i_y, \\ l &= ke, \\ m &= j + l. \end{aligned} \tag{30c}$$

The first block (30a) obtains the evaluation e for $y'(x^0)$. The second block (30b) obtains the evaluation i for $y''(x^0)$. [The special functions f and g are evaluated by calls to DER2(E,1,F) and DER2(E,2,G).] The third block (30c) obtains the evaluation m for $y'''(x^0)$. [The special functions j and k are evaluated by calls to DER1(I,1,J), and DER1(I,2,K).] The evaluations for $y'(x^0)$, $y''(x^0)$ and $y'''(x^0)$ are exact up to round-off error.

Note that the value e for the basic function $x + \log(xy)$ is only calculated once in (30a–c); all remaining calculations consist of simple algebraic and relabeling operations. Also, as is clear from the repetitive form of blocks (30b) and (30c), higher-order power series coefficients could easily be obtained by means of a simple DO loop. The FEED method for obtaining power series coefficients thus appears to provide an attractive alternative to the usual Runge–Kutta and Adams–Moulton methods for solving initial-value problems for ordinary differential equations.

5. CONCLUSION

In the original presentation [1] of the FEED algorithm, three types of calculus subroutines were introduced: subroutine LIN for independent variables, subroutines such as LOGG for functions of one variable and subroutines such as ADD for functions of two variables. The present paper demonstrates that it is both profitable and straightforward to introduce into the FEED library a fourth type of calculus subroutine, the selection subroutine DERK. With this new addition, FEED can *systematically* be applied to a much broader class of functions than previously envisaged.

REFERENCES

1. R. Kalaba, L. Tesfatsion and J.-L. Wang, A finite algorithm for the exact evaluation of higher-order partial derivatives of functions of many variables. *J. math. Analysis Applic.* **12,** 181–191 (1983).
2. R. Wengert, A simple automatic derivative evaluation program. *Communs Ass. comput. Mach.* **7,** 463–464 (1964).
3. R. Kalaba and L. Tesfatsion, Complete comparative static differential equations. *Nonlinear Analysis* **5,** 821–833 (1981).
4. L. B. Rall, *Automatic Differentiation: Techniques and Applications.* Springer, New York (1981).
5. R. Kalaba and A. Tishler, Automatic derivative evaluation in the optimization of nonlinear models. *Rev. Econ. Statist.* **66,** 653–660 (1984).
6. H. Kagiwada, R. Kalaba, N. Rasakhoo and K. Spingarn, *Numerical Derivatives and Nonlinear Analysis.* Plenum Press, New York (1985).
7. A. Wexler, Automatic evaluation of derivatives. Working Paper, Department of Physiology and Biophysics, Univ. of Southern California, Los Angeles, Calif. (June 1985).