17.1 Computation Using Derivative Arrays or Univariate Taylor Series

A. Purpose

This set of subroutines performs computations using arrays of length N+1, where each array contains the value of a function, $f(t)$, and its first N derivatives with respect to t, evaluated at some point, t_0. Such an array can alternatively be regarded as a scaled representation of the first N + 1 coefficients of the Taylor series of $f(t)$ expanded at t_0, since the coefficient of $(t - t_0)^k$ in such a Taylor series is $f^{(k)}(t)/k!$ evaluated at t_0.

This package provides a way of computing values of the first N derivatives of a univariate function that is defined by a sequence of computational steps involving arithmetic and elementary functions, without the need to derive and code expressions for the derivatives.

B. Usage

We shall use t as the generic name of the single independent variable with respect to which all derivatives are defined. We shall use the term W-variable to denote an (N+1)-tuple, consisting of a function value and values of the function's first N derivatives with respect to t, evaluated at some point. Note that the representation of t as a W-variable evaluated at t_0 is the (N+1)-tuple, (t_0, 1, 0, 0, ..., 0).

This package consists of one subroutine, SWSET, for assigning a value to a W-variable, 22 subroutines for doing arithmetic operations and computing elementary functions using W-variables, and three supplementary subroutines, SWCHN, SWRCHN, and SPASCL. These will be described in the following sections:

B.1 SWSET, Assigning a value to a W-variable

B.1.a Program Prototype, Single Precision

INTEGER N

REAL VAL, DERIV, W(≥N+1)

Assign values to N, VAL, and DERIV

CALL SWSET(N, VAL, DERIV, W)

Computed quantities are returned in W().

B.1.b Argument Definitions

N [in] Highest order derivative to be considered. Require $0 \le N \le NMAX$. See Section E for the definition of NMAX.

VAL [in] Value to be assigned to the W-variable, W().

DERIV [in] Value to be assigned as the first derivative of the W-variable, W().

W() [out] Array of length at least N + 1 in which this subroutine will place N + 1 values to represent a W-variable as follows: (VAL, DERIV, 0.0, ..., 0.0). The user should set DERIV = 1.0 to define this variable as the variable, t, with respect to which all differentiation is done. The user can set DERIV = 0.0 to define this W-variable to be a constant, *i.e.* a variable not depending on t.

B.2 Arithmetic and elementary functions using W-variables

In describing the following subroutines, X() and Y() denote input W-variables and Z() denotes an output W-variable. The variables A and I are input variables that are constant relative to t.

In most of these subroutines the output array Z() must occupy storage locations distinct from any of the input data. Exceptions to this rule are SWSUM, SWDIF, SWPRO, SWSUM1, SWDIF1, and SWPRO1. The subroutine SWQUO, which computes $x/y \to z$, permits x and z to occupy the same storage, but y and z must occupy distinct storage.

B.2.a Program Prototype, Single Precision

INTEGER N, I

REAL A, X(≥N+1)**, Y**(≥N+1)**, Z**(≥N+1)

Assign values to N, I, A, X(), and Y(), as appropriate.

Two-argument operations with both arguments depending on t.

CALL SWSUM(N, X, Y, Z)	$x + y \to z$
CALL SWDIF(N, X, Y, Z)	$x - y \to z$
CALL SWPRO(N, X, Y, Z)	$x \times y \to z$
CALL SWQUO(N, X, Y, Z)	$x/y \to z$
CALL SWATN2(N, X, Y, Z)	$\text{atan2}(x, y) \to z$
where for atan2: $-\pi < z \le \pi$ and $\tan(z) = x/y$	

Two-argument operations with only one argument depending on t.

CALL SWSUM1(N, A, Y, Z)	$a + y \to z$
CALL SWDIF1(N, A, Y, Z)	$a - y \to z$
CALL SWPRO1(N, A, Y, Z)	$a \times y \to z$
CALL SWQUO1(N, A, Y, Z)	$a/y \to z$
CALL SWPWRI(N, I, Y, Z)	$y^i \to z$
(See following note.)	

Note: I may be positive, negative, or zero. If I = 0, SWPWRI sets Z(1) = 1.0 and all derivative values of z to 0.0, regardless of the given value of y. It is an error to have $y = 0.0$ when I < 0.

One-argument operations with the argument depending on t.

CALL SWSQRT(N, X, Z)	$\sqrt{x} \to z$
CALL SWEXP(N, X, Z)	$\exp(x) \to z$
CALL SWLOG(N, X, Z)	$\log(x) \to z$
CALL SWSIN(N, X, Z)	$\sin(x) \to z$
CALL SWCOS(N, X, Z)	$\cos(x) \to z$
CALL SWTAN(N, X, Z)	$\tan(x) \to z$
CALL SWASIN(N, X, Z)	$\sin^{-1}(x) \to z$
CALL SWACOS(N, X, Z)	$\cos^{-1}(x) \to z$
CALL SWATAN(N, X, Z)	$\tan^{-1}(x) \to z$
CALL SWSINH(N, X, Z)	$\sinh(x) \to z$
CALL SWCOSH(N, X, Z)	$\cosh(x) \to z$
CALL SWTANH(N, X, Z)	$\tanh(x) \to z$

B.2.b Argument Definitions

N [in] Highest order derivative to be considered. Require $0 \le$ N $\le$ NMAX. See Section E for the definition of NMAX.

A [in] Floating point value that is independent of t.

I [in] Integer value that is independent of t.

X() [in] An input W-variable.

Y() [in] An input W-variable.

Z() [out] An output W-variable.

B.3 SWCHN, application of the chain rule

This subroutine is called by all of the elementary function subroutines. The user will not need to call it directly unless the user is developing a new function subroutine that is not conveniently representable in terms of the available functions.

B.3.a Program Prototype, Single Precision

INTEGER N

REAL X(≥N+1), **F**(≥N+1)

Assign values to N, X(), and F().

CALL SWCHN(N, X, F)

Computed quantities are returned in F().

B.3.b Argument Definitions

N [in] Highest order derivative to be considered. Require $0 \le$ N $\le$ NMAX. See Section E for the definition of NMAX.

X() [in] On entry, X() must contain $x(t_0)$ and derivatives through order N of $x(t)$ with respect to t. X() will be unchanged by this subroutine.

F() [inout] On entry, F() must contain $f(x_0)$ and derivatives through order N of $f(x)$ with respect to x, evaluated at $x_0 = x(t_0)$. On return F() will contain $f(x(t_0))$ and derivatives through order N of $f(x(t))$ with respect to t, evaluated at t_0. Note that the contents of F(1) will remain unchanged.

B.4 SWRCHN, application of the reverse chain rule

This subroutine reverses the action of SWCHN, in the sense that if one were to make the two calls

CALL SWCHN(N, X, F)

CALL SWRCHN(N, X, F)

the first would transform the contents of the array F(), and the second would (if X(2) is nonzero) transform the contents of F() back to the same values (to within computational errors) as before the first call. See Section C for application of SWRCHN to series reversion.

B.4.a Program Prototype, Single Precision

INTEGER N

REAL X(≥N+1), **F**(≥N+1)

Assign values to N, X(), and F().

CALL SWRCHN(N, X, F)

Computed quantities are returned in F().

B.4.b Argument Definitions

N [in] Highest order derivative to be considered. Require $0 \le$ N $\le$ NMAX. See Section E for the definition of NMAX.

X() [in] On entry, X() must contain $x(t_0)$ and derivatives through order N of $x(t)$ with respect to t. X(2), representing dx/dt evaluated at t_0, must be nonzero. The contents of X(1) will not be used by this subroutine. X() will be unchanged by this subroutine.

F() [inout] On entry, F() must contain $f(x(t_0))$ and derivatives through order N of $f(x(t))$ with respect to t, evaluated at $t = t_0$. On return F() will contain $f(x_0)$ and derivatives through order N of $f(x)$ with respect to x, evaluated at $x_0 = x(t_0)$. Note that the contents of F(1) will remain unchanged.

B.5 SPASCL, computation of the "Pascal triangle" of binomial coefficients

B.5.a Program Prototype, Single Precision

INTEGER N

REAL C($\geq$ 1 + (N*(N+1))/2)

Assign a value to N.

CALL SPASCL(N, C)

Computed quantities are returned in C().

B.5.b Argument Definitions

N [in] Highest order derivative to be considered. Require $0 \leq$ N $\leq$ NMAX. See Section E for the definition of NMAX.

C() [out] On return will contain values constituting the "Pascal triangle" of binomial coefficients. For example, if N = 4, the Pascal triangle is

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

SPASCL omits the first element of each row except the first, and thus returns the 11 values: 1, 1, 2, 1, 3, 3, 1, 4, 6, 4, 1.

B.6 Modifications for Double Precision

For double-precision usage, replace the initial S in the name of each subroutine with D, and replace the REAL declarations by DOUBLE PRECISION.

C. Examples and Remarks

As a demonstration problem, consider the following assignments:

$t = 2$
$z_1 = \log(\text{sqrt}(t))$
$z_2 = \exp(2 * z_1)$
$d = z_2 - t$

This should result in $z_2 = t$ and $d = 0$. These quantities and their derivatives through order 3 with respect to t are computed by the program DRDCOMP. The results are shown in ODDCOMP. Note that d and all of its derivatives are zero to machine accuracy, as they should be.

The running time for this package is approximately proportional to N^3. To save time when developing new code using this package, one may run with N = 0 until one is satisfied that the function evaluation is as desired, and then increase N to activate the derivative computation.

Implicit functions and series reversion

Let $[x;t]$ denote a W-variable containing x and all of its derivatives through order N with respect to t, evaluated at t_0. Let $x_0 = x(t_0)$. Suppose one would like to have $[t;x]$. This would represent the inverse function that is defined implicitly by $[x;t]$. When the components of these arrays are interpreted as scaled coefficients of Taylor series the transformation from $[x;t]$ to $[t;x]$ is known as series reversion.

The subroutine SWRCHN can be used to produce $[t;x]$. Note, from the specifications in Section B.4, that given $[x;t]$ and $[f;t]$, SWRCHN transforms $[f;t]$ to $[f;x]$. Formally replacing the symbol f by t, we see that given $[x;t]$ and $[t;t]$, SWRCHN transforms $[t;t]$ to $[t;x]$, thus producing the desired object. Note that the W-variable, $[t;t]$, needed as part of the input, is just (t_0, 1, 0, ..., 0).

D. Functional Description

The "W" in the names of subroutines in this package refers to R. E. Wengert, who, in [1], presented the ideas on which the package is based. Subsequent authors have incorporated this approach into more automated systems [2] that would probably be easier to use than the present package, but such systems present various hurdles regarding acquisition and installation. The present implementation provides the basic capabilities in a highly portable form.

The fundamental idea is simply to provide, for each basic mathematical operation, code that not only performs the operation but propagates derivative values up to the desired order. For example, to support the sine function with derivatives through second degree, we note that if

$$z = \sin y,$$

then

$$z' = y' \cos y,$$

and

$$z'' = y''cosy - y'^2 \sin y,$$

where the primes denote derivatives with respect to an independent variable, t. Clearly these formulas permit one to write a subroutine that can accept y, y', and y'' as input, and produce z, z', and z'' as output.

This approach has been systematized for arbitrary N by appropriate use of the chain rule of differentiation. Our subroutine SWCHN implements the chain rule as

```
do L = 0, N − 1
   call SWPRO(L, F(N+1−L), X(2), F(N+1−L))
enddo
```

Since SWCHN is called by all of the elementary function subroutines, this loop essentially determines the dependence on N of the running time of the whole package. SWPRO is an $O(N^2)$ process and SWCHN is an $O(N^3)$ process. To improve efficiency we have coded the cases of N = 0, 1, 2, 3, and 4 as special cases in SWPRO. Multiplication counts in SWPRO and SWCHN are as follows:

N ⇒	0	1	2	3	4	5	6	7	8	9	10
SWPRO	1	3	7	12	19	27	37	48	61	75	91
SWCHN	0	1	4	11	23	42	69	106	154	215	290

References

1. R. E. Wengert, *A simple automatic derivative evaluation program*, **Comm. ACM 7**, 8 (Aug. 1964) 463–464.

2. Andreas Griewank and George F. Corliss, editors, **Automatic Differentiation of Algorithms, Proceedings of SIAM Workshop on Automatic Differentiation of Algorithms**, SIAM, Philadelphia (Jan. 1991) 353 pages. Contains 31 papers reporting the 1991 state of the art in algorithms and software for automatic differentiation. Includes paper by Lawson specifically relating to the method for series reversion mentioned above in Section C.

3. C. L. Lawson, **Computing Derivatives using W-arithmetic and U-arithmetic**. Internal Computing Memorandum 289, Jet Propulsion Laboratory, Pasadena, CA (Sept. 1971).

E. Error Procedures and Restrictions

E.1 Upper limit on the derivative order, N

Subroutines SWATAN, SWSUM, and SWPRO in this package contain internal arrays whose dimensions depend on NMAX. NMAX is a PARAMETER in each of these subroutines, nominally set to 10. This limits the derivative order argument, N, in all of the subroutines to 10. The user would need to increase NMAX in these three subroutines if higher order derivatives are needed.

E.2 Invalid arguments for derivative computation

The user will likely be accustomed to avoiding sending invalid arguments to the elementary functions, such as a negative argument to the square root. In computing derivatives there are some additional singularities to avoid. Note that the derivative is infinite at zero for the square root, and at ±1 for arcsin and arccosine.

E.3 Error handling

Following is a list of error conditions the package detects and for which error messages are issued. These errors are fatal in the sense that the requested operation cannot be done; however, the default action is to return after issuing an error message. The user can use the MATH77 library subroutine, ERMSET of Chapter 19.2, to alter this action to cause a STOP if desired. Error conditions not on this list, *e.g.*, negative argument in log, will be handled by the usual host system error handler.

Error No. & Program	**Explanation**
1 SWASIN	Infinite derivative when arg = −1 or +1
1 SWACOS	Infinite derivative when arg = −1 or +1
2 SWSQRT	Infinite derivative when arg = 0
3 SWQUO1	Zero divisor
4 SWPWRI	Y**I is infinite when Y = 0 and I < 0
5 SWPRO	Require dimension NMAX ≥ N
6 SWQUO	Require dimension NMAX ≥ N
7 SWQUO	Zero divisor.
8 SPASCL	Require N ≥ 0
9 SWRCHN	Require X(2) ≠ 0.

F. Supporting Information

The source language is ANSI Fortran 77.

All of the double precision entry points require files:

DWCOMP, ERFIN, ERMSG, IERM1, IERV1.

For single precision, file SWCOMP is required in place of file DWCOMP.

Entries

DPASCL	DWACOS	DWASIN	DWATAN
DWATN2	DWCHN	DWCOS	DWCOSH
DWDIF	DWDIF1	DWEXP	DWLOG
DWPRO	DWPRO1	DWPWRI	DWQUO
DWQUO1	DWRCHN	DWSET	DWSIN
DWSINH	DWSQRT	DWSUM	DWSUM1
DWTAN	DWTANH	SPASCL	SWACOS
SWASIN	SWATAN	SWATN2	SWCHN
SWCOS	SWCOSH	SWDIF	SWDIF1
SWEXP	SWLOG	SWPRO	SWPRO1
SWPWRI	SWQUO	SWQUO1	SWRCHN
SWSET	SWSIN	SWSINH	SWSQRT
SWSUM	SWSUM1	SWTAN	SWTANH

Designed by C. L. Lawson, JPL, 1971. Adapted to Fortran 77 for the JPL MATH77 library, Aug. 1987.

DRSWCOMP

```
c      program DRSWCOMP
c>> 1994-10-19 DRSWCOMP Krogh  Changes to use M77CON
c>> 1987-12-09 DRSWCOMP Lawson  Initial Code.
c--S replaces "?": DR?WCOMP, ?WSET, ?VECP, ?WSQRT, ?WLOG, ?WPRO1, ?WEXP,
c--&               ?WDIF, ?WCOMP
c      Demo driver for the SWCOMP package.  Derivative arithmetic.
c     ----------------------------------------------------------------
      integer N, NDIM, NMAX
      parameter(NMAX = 3, NDIM = NMAX+1)
      real              T(NDIM), X1(NDIM), Z1(NDIM), X2(NDIM), Z2(NDIM)
      real              DIFF(NDIM), ONE, TWO
      parameter(ONE = 1.0E0, TWO = 2.0E0)
c     ----------------------------------------------------------------
      print*,'DRSWCOMP..  Demo driver for the SWCOMP package.'
      N = NMAX
c                                          Set T = 2.0
      call SWSET(N, TWO, ONE, T)
      call SVECP(T,N+1,'0 T =')
c                                          Compute Z1 = log(sqrt(T))
      call SWSQRT(N, T, X1)
      call SWLOG(N, X1, Z1)
      call SVECP(Z1,N+1,'0 Z1 = log(sqrt(T)) =')
c
c                                          Compute Z2 = exp(2.0 * Z1)
      call SWPRO1(N, TWO, Z1, X2)
      call SWEXP(N, X2, Z2)
      call SVECP(Z2,N+1,'0 Z2 = exp(2.0 * Z1) =')
c
c                                          Diff = Z2 - T
      call SWDIF(N, Z2, T, DIFF)
      call SVECP(DIFF,N+1,'0 DIFF = Z2 - T =')
      stop
      end
```

ODSWCOMP

```
DRSWCOMP..  Demo driver for the SWCOMP package.

 T =
  1 TO   4     2.000000        1.000000        0.000000        0.000000

 Z1 = log(sqrt(T)) =
  1 TO   4    0.3465736       0.2500000      -0.1250000       0.1250000

 Z2 = exp(2.0 * Z1) =
  1 TO   4     2.000000       0.9999999       0.000000       2.9802322E-08

 DIFF = Z2 - T =
  1 TO   4   -1.1920929E-07 -1.1920929E-07    0.000000       2.9802322E-08
```