17.2 Computation Using Partial Derivative Arrays or Multivariate Taylor Series

A. Purpose

This set of subroutines performs computations in which each variable is represented by its value, and its first and second partial derivatives with respect to N independent variables, evaluated at a specified value of these variables. Such a set of numbers can alternatively be regarded as a scaled representation of the low order coefficients of the multivariate Taylor series of the dependent variable expanded at the evaluation point.

More specifically let $u(\mathbf{x})$ denote a scalar valued function of an N-component argument vector, $\mathbf{x}$. Let u_0, $\mathbf{g}_0$, and H_0 denote, respectively, the value of u, the gradient vector, and the Hessian matrix, evaluated at $\mathbf{x}_0$. Then $\{u_0, \mathbf{g}_0, H_0\}$ is the set of data these programs carry to represent u evaluated at $\mathbf{x}_0$. The Taylor series for u through second order, expanded at $\mathbf{x}_0$, is

$$u(\mathbf{x}) = u_0 + \mathbf{g}_0^t(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^t H_0(\mathbf{x} - \mathbf{x}_0)$$

This package provides a way of computing values of the first and second partial derivatives of a multivariate 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 partial derivatives.

B. Usage

Definition of a U-variable

We shall use $\mathbf{x}$ as the generic name of the N-dimensional independent variable vector with respect to which all partial derivatives are defined. The term U-variable is used to denote a sequence of values, consisting of a function value, values of the function's gradient vector, and optionally the (packed) Hessian matrix, evaluated at some point. We assume the Hessian matrix is symmetric, so only one copy of each symmetric pair of elements needs to be stored. We store the lower triangle of the Hessian matrix by rows. Equivalently, one could regard this as representing the upper triangle by columns. Thus if U() is an array containing a U-variable, $\{u, \mathbf{g}, H\}$, its contents are defined as follows:

$$\begin{aligned} \mathrm{U}(1) &= u \\ \mathrm{U}(1+j) &= g_j \quad = \partial u/\partial x_j,\ j = 1,\ ..., N \\ \mathrm{U}(1+\mathrm{N}+j+i(i-1)/2) &= h_{i,j} = \partial^2 u/\partial x_i \partial x_j, \\ & \qquad i = 1,\ ..., N;\ j = 1,\ ...,\ i. \end{aligned}$$

The required dimension size for a U-variable is $dimu$ = N + 1 if only first partial derivatives will be requested, and $dimu$ = (N +2)(N + 1) / 2 if both first and second partial derivatives are to be computed. For convenience we list the value of $dimu$ **for the second derivative case** for some values of N:

N =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$dimu$ =	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136

Distinguishing property of an independent variable

For i in [1, N] the scalar component, x_i, of the independent variable vector, $\mathbf{x}$, is distinguished by the unique property that all of its first and second partial derivatives are zero except its first partial derivative with respect to x_i, which has the value 1.

The variables N, M1, and M2.

N specifies the number of independent variables, and implicitly specifies the storage layout of components within a U-variable. M1 and M2 specify the lowest and highest order of partial derivatives which the called subroutines are to produce. A program that uses these subroutines must initially call SUSETN (or DUSETN) to set N, M1, and M2.

The subroutines

The subroutines of this package are described in the following sections:

B.1 SUSETN, Assigning values to N, M1, and M2

Subroutine SUSETN must be called prior to calling any of the other subroutines of this package.

B.1.a Program Prototype, Single Precision

INTEGER N, M1, M2

Assign values to N, M1, and M2.

CALL SUSETN(N, M1, M2)

B.1.b Argument Definitions

N [in] N specifies the number of independent variables, and implicitly specifies the storage layout of components within a U-variable.

M1, M2 [in] M1 and M2 specify the lowest and highest order of partial derivatives which the called subroutines are to produce. M1 and M2 must satisfy: $0 \leq$ M1 $\leq$ M2 ≤ 2.

Most commonly one will probably choose to set (M1, M2) = (0,0), (0,1), or (0,2), and leave the setting unchanged throughout a computation. See Section C for discussion of other strategies for setting (M1,M2).

B.2 SUGETN, Fetching values of N, M1, M2, L1, and L2

B.2.a Program Prototype, Single Precision

INTEGER N, M1, M2, L1, L2

CALL SUGETN(N, M1, M2, L1, L2)

Values are returned in N, M1, M2, L1, and L2.

B.2.b Argument Definitions

N, M1, M2 [out] Returns the values that were set on the previous call to SUSETN.

L1, L2 [out] Returns values computed from N, M1, and M2. L1 and L2 are the indices of the first and last locations in a U-variable array that will be subject to change due to the settings of N, M1, and M2.

$$L1 = \begin{cases} 1 & \text{if M1} = 0 \\ 2 & \text{if M1} = 1 \\ N+2 & \text{if M1} = 2 \end{cases}$$

$$L2 = \begin{cases} 1 & \text{if M1} = 0 \\ N+1 & \text{if M1} = 1 \\ 1+N+((N*(N+1))/2) & \text{if M1} = 2 \end{cases}$$

B.3 SUSET, Assigning a value to a U-variable

The integer values N, M1, and M2 must be set by a call to SUSETN before calling SUSET. SUSET will only set partial derivatives of orders M1 through M2.

B.3.a Program Prototype, Single Precision

INTEGER KEY

REAL VAL, U($\geq dimu$)
[$dimu$ = N + 1 or (N + 2)(N + 1)/2, see above.]

Assign values to N, M1, M2, VAL, and KEY.

CALL SUSET(VAL, KEY, U)

Computed quantities are returned in U().

B.3.b Argument Definitions

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

KEY [in] Integer in the range, [0, N]. If KEY = 0, U() is set to represent a variable that is constant relative to the N independent variables, *i.e.*, all of its first and second partial derivatives are set to zero.

If $1 \leq$ KEY $\leq$ N, U() is set to represent the KEYth independent variable, *i.e.*, its KEYth first partial derivative is set to 1.0. All of its other first partial derivatives and all of its second partial derivatives are set to zero.

U() [out] Array in which this subroutine will define a U-variable having the value, VAL, and having partial derivative values as specified by KEY. Only derivative values of orders M1 through M2 are set.

B.4 The computational subroutines, except SUREV

The integer values N, M1, and M2 must be set by a call to SUSETN before calling any of these subroutines. These subroutines will only set partial derivatives of orders M1 through M2. To use these subroutines with M1 $>$ 0 read the remarks about M1 $>$ 0 in Section C.

In describing the following subroutines, U() and V() denote input U-variables and Z() denotes an output U-variable. The variables A and I are input variables that are constant relative to **x**.

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 SUSUM, SUDIF, SUPRO, SUSUM1, SUDIF1, and SUPRO1. The subroutine SUQUO, which computes $u/v \rightarrow z$, permits u and z to occupy the same storage, but v and z must occupy distinct storage.

B.4.a Program Prototype, Single Precision

INTEGER I

REAL A, U($\geq dimu$)**,V**($\geq dimu$)**, Z**($\geq dimu$)
[$dimu$ = N + 1 or (N + 2)(N + 1)/2, see page 1.]

Assign values to I, A, U(), and V(), as appropriate.

Two-argument operations with both arguments depending on x.

CALL SUSUM(U, V, Z)	$u + v \rightarrow z$
CALL SUDIF(U, V, Z)	$u - v \rightarrow z$
CALL SUPRO(U, V, Z)	$u \times v \rightarrow z$
CALL SUQUO(U, V, Z)	$u/v \rightarrow z$
CALL SUATN2(U, V, Z)	atan2$(u, v) \rightarrow z$

where for atan2: $-\pi < z \leq \pi$ and $\tan(z) = u/v$

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

CALL SUSUM1(A, V, Z)	$a + v \to z$
CALL SUDIF1(A, V, Z)	$a - v \to z$
CALL SUPRO1(A, V, Z)	$a \times v \to z$
CALL SUQUO1(A, V, Z)	$a/v \to z$
CALL SUPWRI(I, V, Z)	$v^i \to z$
(See following note.)	

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

One-argument operations with the argument depending on x.

CALL SUSQRT(U, Z)	$\sqrt{u} \to z$
CALL SUEXP(U, Z)	$\exp(u) \to z$
CALL SULOG(U, Z)	$\log(u) \to z$
CALL SUSIN(U, Z)	$\sin(u) \to z$
CALL SUCOS(U, Z)	$\cos(u) \to z$
CALL SUTAN(U, Z)	$\tan(u) \to z$
CALL SUASIN(U, Z)	$\text{asin}(u) \to z$
CALL SUACOS(U, Z)	$\text{acos}(u) \to z$
CALL SUATAN(U, Z)	$\text{atan}(u) \to z$
CALL SUSINH(U, Z)	$\sinh(u) \to z$
CALL SUCOSH(U, Z)	$\cosh(u) \to z$
CALL SUTANH(U, Z)	$\tanh(u) \to z$

B.4.b Argument Definitions

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

I [in] Integer value that is independent of **x**.

U() [in] An input U-variable. Must contain defined values for derivatives of orders 0 through M2.

V() [in] An input U-variable. Must contain defined values for derivatives of orders 0 through M2.

Z() [inout] If M1 > 0, this array must contain defined values of derivatives of orders 0 through M1 − 1 on entry. On return it will also contain computed values of derivatives of orders M1 through M2. If M1 = 0, no input values are required in Z().

B.5 SUREV, Series Reversion or Function Inversion

Let u_i, $i = 1, ..., n$, be n functions of an n-vector, **t**. Suppose values of the u_i's and their first and second partial derivatives with respect to the components of **t** are known for a particular value of **t**. Then, if the Jacobian matrix of the transformation is nonsingular at this point, one can regard the t_j's as functions of the u_i's in some neighborhood of this point in u-space. In this situation this subroutine can compute values of the first and second partial derivatives of the t_j's with respect to the u_i's at this point.

Given:

$$t_j \text{ in TU}(1,\ j),\ j = 1,\ ...,\ n, \tag{1}$$

$$u_i \text{ in UT}(1,\ i),\ i = 1,\ ...,\ n, \tag{2}$$

$$\partial u_i/\partial t_j \text{ in UT}(1 + j,\ i),\ j = 1,\ ...,\ n;\ i = 1,\ ...\ n, \tag{3}$$

$$\partial^2 u_i/\partial t_j \partial t_k \text{ in UT}(1 + n + k + j(j-1)/2,\ i), \tag{4}$$
$$j = 1,\ ...,\ n;\ k = 1,\ ...,\ j;\ i = 1,\ ...,\ n.$$

If the Jacobian matrix with elements $\partial u_i/\partial t_j$ is nonsingular, this subroutine will compute the first and second partial derivatives of the t_j's with respect to the u_i's and store them as

$$\text{TU}(1 + i, j) = \partial t_j/\partial u_i,\ i = 1,\ ...,\ n;\ j = 1,\ ...,\ n, \tag{5}$$

$$\text{TU}(1 + n + k + i(i-1)/2,\ j) = \partial^2 t_j/\partial u_i \partial u_k, \tag{6}$$
$$i = 1,\ ...,\ n;\ k = 1,\ ...,\ i;\ j = 1,\ ...,\ n,$$

The integer values N, M1, and M2 must be set by a call to SUSETN before calling SUREV. N gives the value of n for Eqs.(1–6) above. Require $0 \le \text{M1} \le \text{M2} \le 2$.

If M2 = 0, SUREV returns with no action.

If M2 = 1, SUREV only computes Eq.(5), and it is not necessary to provide the data of Eq.(4).

If M2 = 2 and M1 = 0 or 1, SUREV computes Eqs.(5–6).

If M2 = 2 and M1 = 2, it is assumed that the assignment of Eq.(5) has already been done, and thus SUREV only computes Eq.(6).

B.5.a Program Prototype, Single Precision

INTEGER IWORK(N)**, LDIM**

REAL RCOND, TU(LDIM, ≥N)**, UT**(LDIM, ≥N)**, WORK**(≥ 3 × N^2) [LDIM ≥ $dimu$ = N + 1 or (N + 2)({$textN$ + 1)/2, see page 1.]

Assign values to LDIM, UT() and part of TU().

CALL SUREV(UT, TU, LDIM, RCOND, IWORK, WORK)

Computed quantities are returned in RCOND and TU().

B.5.b Argument Definitions

UT(,) [in] Must contain the data of Eqs.(2–4) on entry, except as noted above for certain values of M1 and M2.

TU(,) [inout] Must contain the data of Eq.(1) on entry, and will have the data of Eqs.(5–6) assigned by SUREV, except as noted above for particular values of M1 and M2.

LDIM [in] Leading dimension for the arrays UT(,) and TU(,). Require LDIM $\geq$ *dimu*, where *dimu* = N + 1 or (N + 2)(N + 1)/2, see page 1.

RCOND [out] Estimate of the reciprocal condition number of the Jacobian matrix with elements $\partial u_i/\partial t_j$, given in UT(). Only computed when M1 $\leq$ 1 and M2 $\geq$ 1. Is in the range [0.0, 1.0].

A well conditioned problem has values near 1.0. Near zero means badly conditioned. Equal to zero means singular, in which case the subroutine cannot complete its computation.

IWORK() [scratch] Integer work space of size at least N.

WORK() [scratch] Floating-point work space of size at least $3 \times N^2$.

B.6 Modifications for Double Precision

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

C. Examples and Remarks

C.1 Example

As a demonstration problem, consider the following formulae for transforming a set of 3-dimensional rectangular coordinates (x, y, z) to spherical coordinates (r, φ, θ) and the inverse transformation formulae:

$$
\begin{aligned}
s &= \sqrt{x^2 + y^2} \\
r &= \sqrt{s^2 + z^2} \\
\varphi &= \text{atan2}(y,\ x) \\
\theta &= \tan^{-1}(z/s) \\
x &= r\ \cos\ \varphi\ \cos\ \theta \\
y &= r\ \sin\ \varphi\ \cos\ \theta \\
z &= r\ \sin\ \theta
\end{aligned}
$$

The demonstration program, DRSUCOMP, performs this mapping from (x, y, z) to (r, φ, θ) with computation of all of the first and second partial derivatives of r, φ, and θ with respect to x, y, and z. As a check on the computation the program then transforms back to (x, y, z). Results are shown in ODSUCOMP. Note that the final (x, y, z) agrees with the initial (x, y, z) in the values and the first and second partial derivatives.

To demonstrate series reversion (or function inversion) the program assigns the values of (r, φ, θ) and its first and second partial derivatives with respect to (x, y, z) to the array UT(,), and then assigns the values of (x, y, z) to TU(1, 1:3). It then uses SUREV to compute the first and second partial derivatives of (x, y, z) with respect to (r, φ, θ), storing these in TU(2:10, 1:3). This computation is checked by computing the same quantities in a different way.

C.2 Omitting derivative computation

To save time when developing new code using this package, one may run with (M1, M2) = (0, 0) until one is satisfied that the function evaluation is as desired, and then increase M2 to activate the derivative computation.

C.3 Setting M1 > 0

There are some algorithms, such as for optimization, in which one does not need to compute partial derivatives at every point at which the function is evaluated. Depending on how significant efficiency is in a particular application, one may wish to consider methods of separating the function and derivative computation using this package. One can compute only the function value by setting (M1, M2) = (0, 0), or one can compute the function value and first partial derivatives by setting (M1, M2) = (0, 1).

If one has computed the function value at an argument value, **x**, with (M1, M2) = (0, 0), and has kept all intermediate quantities in distinct storage locations, then one can repeat the sequence of subroutine calls with (M1, M2) = (1, 1) to compute the first partial derivatives at the same point, **x**, without the function value being recomputed. In general, computation with M1 > 0 is only valid if all *in* and *inout* arrays in each subroutine call contain values of all derivatives of orders 0 through M1 − 1 computed previously with the same **x**.

D. Functional Description

This U-computation package is based on the ideas presented in [1]. See the description of W-computation in Chapter 17.1 for a summary of these ideas. Whereas the W-computation package generalizes the order of differentiation, this U-computation package generalizes the number of independent variables.

Let $f()$ be a scalar-valued function of the scalar variable u, and use primes to denote differentiation with respect to u. Let u depend on the N-vector **x** and denote partial derivatives of u with respect to x_i by appropriate subscripts. This package computes first and second partial derivatives of $f()$ using the formulae:

$$
\begin{aligned}
\partial f(u)/\partial x_i &= f'(u)u_i \\
\partial^2 f(u)/\partial x_i \partial x_j &= f'(u)u_{i,j} + f''(u)u_i u_j
\end{aligned}
$$

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

N must be positive. M1 and M2 must satisfy $0 \leq$ M1 $\leq$ M2 ≤ 2. See the discussion in Section C for cautions regarding usage with M1 > 0. These conditions on N, M1, and M2 are not tested by the subroutines and their violation may cause unpredictable actions.

E.1 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.2 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 SUASIN	Infinite derivative when arg $= -1$ or $+1$
1 SUACOS	Infinite derivative when arg $= -1$ or $+1$
2 SUSQRT	Infinite derivative when arg $= 0$
3 SUQUO1	Zero divisor
4 SUPWRI	U^M is infinite when U $= 0$ and M < 0
5 SUQUO	Zero divisor
6 SUREV	Singular Jacobian matrix

F. Supporting Information

The source language is ANSI Fortran 77. All program units reference COMMON blocks /UCOM1/ and /UCOM2/.

All of the double precision entry points except DUREV require files:

DUCOMP, ERFIN, ERMSG, IERM1, and IERV1.

All of the single precision entry points except SUREV require files:

SUCOMP, ERFIN, ERMSG, IERM1, and IERV1.

DUREV requires: DASUM, DAXPY, DDOT, DGECO, DGEFA, DGEI, DSCAL, DSWAP, DUREV, ERFIN, ERMSG, and IDAMAX.

SUREV requires: SASUM, SAXPY, SDOT, SGECO, SGEFA, SGEI, SSCAL, SSWAP, SUREV, ERFIN, ERMSG, and ISAMAX.

Entries

DUACOS	DUASIN	DUATAN	DUATN2
DUCOS	DUCOSH	DUDIF	DUDIF1
DUEXP	DUGETN	DULOG	DUPRO
DUPRO1	DUPWRI	DUQUO	DUQUO1
DUREV	DUSET	DUSETN	DUSIN
DUSINH	DUSQRT	DUSUM	DUSUM1
DUTAN	DUTANH	SUACOS	SUASIN
SUATAN	SUATN2	SUCOS	SUCOSH
SUDIF	SUDIF1	SUEXP	SUGETN
SULOG	SUPRO	SUPRO1	SUPWRI
SUQUO	SUQUO1	SUREV	SUSET
SUSETN	SUSIN	SUSINH	SUSQRT
SUSUM	SUSUM1	SUTAN	SUTANH

Designed by C. L. Lawson, JPL, 1971. Adapted to Fortran 77 for the JPL MATH77 library, Aug. 1987. Added SUREV/DUREV, February 1992. Added SUSETN, DUSETN, SUGETN, and DUGETN August 1994.

DRSUCOMP

```
c      program DRSUCOMP
c>> 1994-10-19 DRSUCOMP Krogh  Changes to use M77CON
c>> 1994-08-04 DRSUCOMP CLL New subroutine: SUSETN
c>> 1992-02-17 CLL
c>> 1990-12-13 CLL Added demo of SUREV.
c>> 1987-12-09 DRSUCOMP Lawson  Initial Code.
c      Demo driver for the SUCOMP package, including SUREV.
c      The SUCOMP package computes partial derivatives.
c      SUREV does series reversion involving 1st and 2nd partial
c      derivatives of N functions of N variables.
c     ----------------------------------------------------------------
c--S replaces "?": DR?UCOMP, ?UCOMP, ?UREV, ?USETN, ?USET, ?UATN2
c--    &          ?UPRO, ?USUM, ?USQRT, ?UQUO, ?UATAN, ?UCOS, ?USIN, ?COPY
c     ----------------------------------------------------------------
      integer I, LDIM, NMAX
      parameter(NMAX = 3, LDIM = ((NMAX+2)*(NMAX+1))/2)
      integer IWORK(NMAX)
      real              X(LDIM), Y(LDIM), Z(LDIM)
      real              R(LDIM), PHI(LDIM), THETA(LDIM)
      real              SSQ(LDIM), S(LDIM), TEMP(LDIM)
      real              X2(LDIM), Y2(LDIM), Z2(LDIM)
      real              XSQ(LDIM), YSQ(LDIM), ZSQ(LDIM)
      real              XVAL, YVAL, ZVAL, RSQ(LDIM), ZBYS(LDIM)
      real              SP(LDIM), CP(LDIM), ST(LDIM), CT(LDIM)
      real              WORK(NMAX,NMAX,3), RCOND
      real              UT(LDIM,NMAX), TU(LDIM,NMAX)
      real              X1(LDIM), X3(LDIM), T1(LDIM), T2(LDIM), T3(LDIM)
      parameter(XVAL = 0.1E0, YVAL = 0.2E0, ZVAL = 0.3E0)
c     -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
c                                     Set N, M1, and M2.
      call SUSETN(NMAX, 0, 2)
c                                     Initialize independent variables.
      call SUSET(XVAL,1,X)
      call SUSET(YVAL,2,Y)
      call SUSET(ZVAL,3,Z)
      print'(1x,a)',
     * 'DRSUCOMP..  Demo driver for the SUCOMP package.',
     * 'This demo first transforms (x,y,z) to (r,theta,phi),',
     * 'and then transforms back, including computation of',
     * '1st and 2nd partial derivatives.'
      write(*,'(1x/8x,10a)') '  VALUE','     D1','     D2','     D3',
     *    '    D11','    D21','    D22','    D31','    D32','    D33'
      write(*,'(1x/(1x,a,10f7.3))')
     *   '    X =',X,'    Y =', Y,'    Z =',Z
c
c                                  Transform from (x,y,z) to (r,theta,phi)
      call SUATN2(Y, X, PHI)
      call SUPRO(X, X, XSQ)
      call SUPRO(Y, Y, YSQ)
      call SUPRO(Z, Z, ZSQ)
      call SUSUM(XSQ, YSQ, SSQ)
      call SUSUM(SSQ, ZSQ, RSQ)
      call SUSQRT(RSQ, R)
      call SUSQRT(SSQ, S)
      call SUQUO(Z, S, ZBYS)
      call SUATAN(ZBYS, THETA)
```

```
      write(*,'(1x/(1x,a,10f7.3))')
     *   '     R =',R,'   PHI =',PHI,'THETA =',THETA
c
c                              Transform back from (r,theta,phi) to (x,y,z)
      call SUCOS(PHI, CP)
      call SUCOS(THETA, CT)
      call SUPRO(CP, CT, TEMP)
      call SUPRO(TEMP, R, X2)
      call SUSIN(PHI, SP)
      call SUPRO(SP, CT, TEMP)
      call SUPRO(TEMP, R, Y2)
      call SUSIN(THETA, ST)
      call SUPRO(ST, R, Z2)
      write(*,'(1x/(1x,a,10f7.3))')
     *   '     X =',X2,'     Y =',  Y2,'     Z =',Z2
c
c                              Set data to call SUREV.
c
      call SCOPY(LDIM, R,1, UT(1,1),1)
      call SCOPY(LDIM, PHI,1, UT(1,2),1)
      call SCOPY(LDIM, THETA,1, UT(1,3),1)
      TU(1,1) = X(1)
      TU(1,2) = Y(1)
      TU(1,3) = Z(1)
      call SUREV( UT, TU, LDIM, RCOND, IWORK, WORK)

      write(*,'(a)') ' ',
     * ' To demo SUREV we store (R, PHI, THETA) including',
     * ' the first and second derivatives w.r.t. (X,Y,Z) into UT(),',
     * ' and set TU() = (X, Y, Z).',
     * ' Then compute (TU1,TU2,TU3) using SUREV.',
     * ' For comparison compute (T1,T2,T3) using the known functional',
     * ' definition of (X, Y, Z) as a function of (R, PHI, THETA).'

      write(*,'(1x/(1x,a,10f7.3))')
     *   ' TU1 =',(TU(I,1),I=1,LDIM),
     *   ' TU2 =',(TU(I,2),I=1,LDIM),
     *   ' TU3 =',(TU(I,3),I=1,LDIM)
c
c             For comparison set X1, X2, X3, and transform them to
c             T1, T2, T3.  These are the same operations as
c             transforming from (r,theta,phi) to (x,y,z).
c
      call SUSET(R(1),1,X1)
      call SUSET(PHI(1),2,X2)
      call SUSET(THETA(1),3,X3)
c
      call SUCOS(X2, CP)
      call SUCOS(X3, CT)
      call SUPRO(CP, CT, TEMP)
      call SUPRO(TEMP, X1, T1)
      call SUSIN(X2, SP)
      call SUPRO(SP, CT, TEMP)
      call SUPRO(TEMP, X1, T2)
      call SUSIN(X3, ST)
      call SUPRO(ST, X1, T3)
      write(*,'(1x/(1x,a,10f7.3))')
     *   '    T1 =',T1,'    T2 =',  T2,'    T3 =',T3
      end
```

ODSUCOMP

```
DRSUCOMP..  Demo driver for the SUCOMP package.
This demo first transforms (x,y,z) to (r,theta,phi),
and then transforms back, including computation of
1st and 2nd partial derivatives.
```

	VALUE	D1	D2	D3	D11	D21	D22	D31	D32	D33
X =	0.100	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Y =	0.200	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Z =	0.300	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
R =	0.374	0.267	0.535	0.802	2.482	−0.382	1.909	−0.573	−1.145	0.955
PHI =	1.107	−4.000	2.000	0.000	16.000	12.000	−16.000	0.000	0.000	0.000
THETA =	0.930	−0.958	−1.917	1.597	−6.297	6.571	3.559	0.913	1.825	−6.845
X =	0.100	1.000	−0.000	−0.000	−0.000	0.000	0.000	−0.000	0.000	0.000
Y =	0.200	0.000	1.000	−0.000	−0.000	0.000	0.000	0.000	−0.000	0.000
Z =	0.300	0.000	0.000	1.000	−0.000	−0.000	−0.000	0.000	0.000	−0.000

```
To demo SUREV we store (R, PHI, THETA) including
the first and second derivatives w.r.t. (X,Y,Z) into UT(),
and set TU() = (X, Y, Z).
Then compute (TU1,TU2,TU3) using SUREV.
For comparison compute (T1,T2,T3) using the known functional
definition of (X, Y, Z) as a function of (R, PHI, THETA).
```

TU1 =	0.100	0.267	−0.200	−0.134	−0.000	−0.535	−0.100	−0.359	0.268	−0.100
TU2 =	0.200	0.535	0.100	−0.268	−0.000	0.267	−0.200	−0.717	−0.134	−0.200
TU3 =	0.300	0.802	0.000	0.224	0.000	−0.000	0.000	0.598	0.000	−0.300
T1 =	0.100	0.267	−0.200	−0.134	0.000	−0.535	−0.100	−0.359	0.268	−0.100
T2 =	0.200	0.535	0.100	−0.268	0.000	0.267	−0.200	−0.717	−0.134	−0.200
T3 =	0.300	0.802	0.000	0.224	0.000	0.000	0.000	0.598	0.000	−0.300