12.2 Multi-Dimensional Table Look Up, Interpolation, and Differentiation

A. Purpose

Given a multi-dimensional table of independent variable values and the corresponding dependent variable values, this subroutine finds the points in the table closest to a given value of the independent variable vector and uses these points to interpolate for the corresponding value of the dependent variable. Error estimates, different look up methods, and the computation of derivative information are available. Ragged tables, a generalization of having known data defined on a grid, are supported.

B. Usage

B.1 Program Prototype, Single Precision

INTEGER NDIM, NTAB($\geq$ see below)**,**
NDEG($\geq$NDIM)**, LUP**($\geq$NDIM)**, IOPT**($\geq k$)
[k depends on options used (≥ 3).]

REAL X($\geq$NDIM)**, Y, XT**($\geq$ see below),
YT($\geq$ see below)**, EOPT**($\geq$IOPT(2))

CALL SILUPM(NDIM, X, Y, NTAB, XT, YT, NDEG, LUP, IOPT, EOPT)

B.2 Argument Definitions

Data tables can be organized in two distinct ways – the grid method and the ragged table method. The specifications of NTAB(), XT(), and YT() are given in this section for the (simpler) grid method. See Section B.3 for the ragged table method.

NDIM [in] Number of dimensions for the independent variable, $1 \leq$ NDIM ≤ 10.

X() [in] Independent vector value, $\mathbf{x}$, where value of the interpolant is desired, $\mathbf{x} = (x_1, x_2, ..., x_{\text{NDIM}})$.

Y [out] Value of interpolant.

NTAB() [inout] For the grid method, NTAB(i) gives the number of points in the table data with respect to the i^{th} dimension for $1 \leq$ i $\leq$ NDIM. NTAB must have a declared dimension $\geq 2\times$ NDIM + 1. On the initial entry NTAB(NDIM+1) must be 0. On the initial entry (signaled by NTAB(NDIM+1) being 0) the subroutine will change the contents of NTAB(NDIM+1) and store values in NTAB(NDIM+2:2×NDIM+1). The user must not alter the contents of NTAB() while continuing to do interpolations in the same table. See Section B.3 for ragged tables.

XT() [in] Array of independent variable values for the tables. For the grid method the user may store the NTAB(1) tabular values of x_1 in the first NTAB(1) locations of XT(), followed immediately by the NTAB(2) tabular values of x_2 in the next NTAB(2) locations, etc. For each variable its values must be either in nondecreasing or nonincreasing order, but this is not checked for. Consecutive values listed for one variable may be equal, but this has a special meaning as described in Chapter 12.1.

For each variable that is to be given at an equally spaced sequence of values one can use an alternative method of specifying the values, *i.e.*, one can just list two values giving respectively the first value and the increment between values. If this is done for the i^{th} variable it must be signaled by setting LUP(i) = 3. When the specification of the i^{th} variable is abbreviated in this way the specification values for the $i+1^{st}$ variable (if any) must begin in the immediately following location in XT().

If no entries in LUP() are set to 3 the minimal dimension for XT() is $\sum_{i=1}^{\text{NDIM}}$ NTAB(i). For each i with LUP(i) = 3, replace NTAB(i) by 2 in this sum for determining the dimension. See Section B.3 for ragged tables.

YT() [in] Array of dependent variable values for the tables. For the grid method the number of values to be given in YT() and the minimal dimension for YT() is the product of the values of NTAB(i) for i = 1, ..., NDIM. The ordering of the y values must be such that the index of the last independent variable is advanced most rapidly. For example, when NDIM = 2 the order of the values must be

$$((y(x_{1,i}, x_{2,j}),\ j=1,\ NTAB(2)),\ i=1, NTAB(1))$$

and when NDIM = 3 the order of the values must be

$$(((y(x_{1,i}, x_{2,j}, x_{3,k}),\ k=1,\ NTAB(3)),\\ j=1,\ NTAB(2)),\ i=1,\ NTAB(1)).$$

See Section B.3 for ragged tables.

NDEG() [in] NDEG(i) defines the nominal degree of the polynomial to be used in the i^{th} dimension. The definition for NDEG(i) is like that of NDEG in the one-dimensional case; see the write-up for SILUP in Chapter 12.1.

LUP() [inout] LUP(i) defines the type of look up method for the i^{th} dimension exactly as LUP does for the one-dimensional case in Chapter 12.1, except that the value 4 should not be used here..

IOPT() [inout] IOPT(1) is used to return a status as follows:

- 0 Normal return, no exceptional conditions.
- 1 X was outside the domain of the table, extrapolation used.
- 2 Available table values were so few that this restricted the degree of the polynomial.
- −1 Error estimate is greater than requested error.
- −2 Bad value for NDIM.
- −3 Bad value for LUP(i).
- −4 Bad specification for a ragged table.
- −5 Ragged table does not start with a 0.
- −6 Bad value inside a ragged table.
- −7 Bad value at end of ragged table.
- −8 Bad option index.
- −9 In some dimension, the first and last XT values are equal.
- −10 Too many derivatives requested.
- −11 Bad value for number of derivatives in some dimension.
- −12 Problem with storage use in EOPT.
- <−20 Had an error in SILUP. The value returned is (the value set by SILUP) − 20.

IOPT(2) gives the dimension of EOPT. In addition to the first location for the error estimate, and the locations in EOPT used for options, SILUPM requires a contiguous block of storage in EOPT of length $2 \times \left[2 \times \text{NDIM} + \sum_{i=1}^{\text{NDIM}-1} \text{NDEG}(i)\right] +$ any additional space required for temporary derivative storage (see option 3 below). If you would like a message detailing memory requirements, run the program with IOPT(2) = 0, and everything else as it would ordinarily be set.

Starting with IOPT(3), options are specified by integers in the range 0 to 6, followed in some cases by integers providing argument(s) for the option. Each option, with its arguments if any, is followed in IOPT by the next option or by a 0. If an option index is specified more than once, only the last specification is used.

- 0 End of the option list; this must always be the last option specified in IOPT.
- 1 An error estimate is to be returned in EOPT(1).
- 2 (Arguments: $K2_1$, ..., $K2_{\text{NDIM}}$)$K2_i$ gives the polynomial degree to use when extrapolating with respect to the i^{th} dimension. Documentation for SILUP (Chapter 12.1) gives the default.
- 3 (Arguments: K3, L3, $M3_1$, ..., $M3_{NDIM}$) This requests computation of derivatives. Let $D(i_1, i_2, ..., i_{\text{NDIM}})$ denote the result of differentiating P, the interpolating polynomial, i_1 times with respect to x_1, i_2 times with respect to x_2, ..., i_{NDIM} times with respect to x_{NDIM}, all divided by $i_1!i_2!...i_{NDIM}!$. The i_j satisfy the restrictions $\Sigma i_j \leq$ L3 and $0 \leq i_j \leq$ M3$_j$. Require $0 \leq$ M3$_j$ $\leq$ NDEG(j). Subject to these restrictions, the D's are stored starting at EOPT(K3) in lexicographic order of i_{NDIM}, ..., i_1. (Note that for XT, YT, and NTAB the lexicographic order is in terms of putting i_1 first in the list.) Thus, for example, if NDIM = 2, L3 = 2, and $M3_1 = M3_2 = 1$, then EOPT(K3) = $\partial P/\partial x_1$, EOPT(K3+1) = $\partial P/\partial x_2$, and EOPT(K3+2) = $\partial^2 P/\partial x_1 \partial x_2$. Note that if L3 had been 1, nothing would have been stored in EOPT(K3+2). The Lexicographic ordering here was (0,1), (1,0), (1,1), where (0,1) indicates the 0^{th} partial with respect to x_2, and the first partial with respect to x_1. To clarify what we mean by a lexiographic ordering, if one is getting all the derivatives up to second with respect to every variable, in three dimensions, the order looks like (0,0,1), (0,0,2), (0,1,0), (0,1,1), (0,1,2), (0,2,0), (0,2,1), (0,2,2), (1,0,0), (1,0,1), (1,0,2), (1,1,0), ...(2,2,0), (2,2,1), (2,2,2). If one only want derivative up to a total derivative of order 2, delete from the list, those where the sum of the numbers is > 2.

  Extra storage in EOPT is required when this option is used, as mentioned in the description of IOPT(2) above. Let D_j denote the space for the total number of derivatives that must be computed in dimension j to get the final D_1 derivatives in dimension 1 (with 0^{th} derivatives being counted in the space). We do not have an explicit formula for the D_j, but a recurrence is given at the end of Functional Description below. There must, of course, be $D_1 - 1$ locations available starting at K3 for storing the final result. In addition, $\sum_{j=2}^{\text{NDIM}}(\text{NDEG}(j-1)+2)(D_j - 1)$ locations are required for temporary storage. If you would like a suggestion on how much memory to set aside and what value to assign to K3, set K3 = 0 and such a suggestion will be printed. If K3 is set to −1 and IOPT(2) is large enough, a value will be selected for K3, the location in IOPT containing K3 will be set to this value, and computation will be done as if this value had been assigned in the first place.
- 4 (Argument K4) The absolute and relative

errors expected in YT entries are specified in EOPT(K4) and EOPT(K4+1) respectively. The values provided here are used in estimating the error in the interpolation. An error estimate is returned in EOPT(1).

5 (Argument K5) Do the interpolation to the accuracy requested by the absolute error tolerance specified in EOPT(K5). An attempt is made to keep the final error $<$ EOPT(K5). Standard polynomial interpolation is done, but here NDEG() gives the maximal degree polynomial to use in the interpolation. If EOPT(K5) ≤ 0, IOPT(1) is not set to -1, and no error message is generated due to an unsatisfied accuracy request. An error estimate is returned in EOPT(1).

6 (Argument K6) Do not use a point in the interpolation if the value of the dependent variable at that point is equal to EOPT(K6). This option can simplify the specification of tables, if some of the points that would naturally be included do not have good data associated with them. This option only affects interpolations done in the last dimension since such interpolations are the only ones that are based directly on YT. Inaccurate results are liable to result if valid data points are too widely separated when doing an interpolation in the last dimension. Thus for example, in a 2-dimensional grid, if bad data points tend to occur in runs that correspond to the same value of the first variable, one would be better off interchanging these variables.

EOPT() [inout] Array used as follows.

EOPT(1) [out] contains an estimate of the error in the interpolation if an error estimate has been requested.

EOPT(2:IOPT(2)) [in or out] for use by options 3–6, and for temporary storage.

B.3 Ragged Tables Defined, with Examples for all Possible 2 and 3-Dimensional Tables

The description above gives all the information necessary to interpolate data on a grid. Tables 1 and 2 below illustrate the setting of NDIM, NTAB(), XT(), and YT() for 2-D and 3-D grid cases respectively. The values shown following the symbol " $\Rightarrow$ " following NTAB() in these tables are the values the subroutine computes and inserts into the NTAB() array on the initializing call. Generally the user need not be concerned with these values.

Here we define ragged tables and specify how to set NTAB(), XT(), and YT() for this more general case. The capability to interpolate ragged tables provides for more general tables than simple grids, but requires less storage and allows a simpler interpolation algorithm than would be needed for general scattered data. Information in this section is recommended as preparation if one intends to understand the details in Section D.

If there is just one set of x_i values where values of y are specified then x_i is said to be a grid variable, whereas if the set of x_i values where values of y are specified is different depending on the selection of values of one or more of the variables with indexes $j < i$, x_i is called a ragged variable. This subroutine requires the first variable, x_1, to be a grid variable and further requires that all grid variables come before all ragged variables, if any. Thus we can define an integer $n_g \geq 1$ to denote the number of grid variables, and the variables x_i with $i \leq n_g$ will be grid variables while the remaining variables, if any, will be ragged variables. The table is regarded as a grid if all variables are grid variables, and as a ragged table otherwise.

If x_i is a grid variable, NTAB(i) must indicate the number of tabular values for x_i, as previously described. If x_i is a ragged variable, NTAB(i) must be set to a negative value, indicating that value sets for x_i depend on the values of the variables x_1 through $x_{|\mathrm{NTAB}(i)|}$. The value of NTAB(i) is required to be $-(i-1)$ if $i <$ NDIM or if $i =$ NDIM and x_{NDIM} is not the only ragged variable, whereas if $i =$ NDIM and x_{NDIM} is the only ragged variable, NTAB(i) can be any value from -1 through $-$(NDIM $-$ 1).

As a consequence of these rules there are only two different possibilities with NDIM = 2 and four different possibilities with NDIM = 3. Examples of each of these six cases are given in Tables 1 to 6, whose characteristics are summarized in the following list:

Table	NDIM	NTAB(1:NDIM)	Description
1	2	n_1, n_2	2-D grid
2	3	n_1, n_2, n_3	3-D grid
3	2	n_1, -1	2-D ragged
4	3	n_1, n_2, -2	3-D ragged
5	3	n_1, -1, -2	3-D ragged
6	3	n_1, n_2, -1	3-D ragged

In the 2-D ragged table, the x_2 values must be specified depending on x_1. In the 3-D tables, Tables 4 and 6 have x_2 values on a grid, and thus these values need not be specified. In Table 5, the values of x_2 depend on the value of x_1. In Tables 4 and 5, the x_3 values depend on both x_1 and x_2 values. In Table 6, the x_3 values depend only on x_1.

Note that a 3-D table with x_2 and x_3 values both depending only on x_1 is not allowed.

B.3.a Specification of NTAB() for a ragged table

One must set NTAB(1:NDIM) as described above, and set NTAB(NDIM+1) = NTAB(3×NDIM+1) = 0. NTAB(NDIM+2:3×NDIM) need not be initialized.

Let r denote the index of the first ragged variable. The sizes of the value sets for x_r must be stored in NTAB() consecutively beginning at NTAB(3×NDIM+2). There must be one size for each possible combination of values of the variables indexed from 1 through |NTAB(r)|. These sizes must be ordered in lexicographic order of the indexes of the values of the variables on which they depend.

As an example, consider Table 3 where the first ragged variable is x_2 and it necessarily depends only on x_1. The variable x_1 has four values, so four sizes must be given. These are set as 3,2,3,2 in NTAB(8:11).

As a more complicated example, consider Table 4 where the first ragged variable is x_3 and it depends on x_1 and x_2. Since x_1 has three values and x_2 has two values the number of combinations of values of x_1 and x_2 is six, and therefore six sizes must be given for value sets for x_3. These are given as 2,3,4,3,2,3 in NTAB(11:16). This ordering results from first fixing x_1 at its first value and running x_2 through its two values, then fixing x_1 at its second value and again running x_2 through its two values, and finally fixing x_1 at its third value and again running x_2 through its two values.

If there are no more ragged variables the next available location in NTAB() must be set to −1 to signal the end of information in NTAB(). Otherwise this next available location must be set to zero and the sizes for the next ragged variable must be stored consecutively in the following locations.

Our only example having more than one ragged variable is Table 5. The second ragged variable is x_3 and it depends on x_1 and x_2, with x_2 itself being ragged. To count the number of sizes needed for x_3 one must make use of the sizes already set for x_2. Thus with the first value of x_1 there are 3 values for x_2, while the second and third values of x_1 each have 2 associated values of x_2. Thus there is a total of 7 possible combinations of values of x_1 and x_2, and so seven sizes must be given for x_3. These are given as 2,3,2,3,2,2,3 in NTAB(15:21).

B.3.b Specification of XT() for a ragged table

Value sets for the grid variables must be entered into XT() as previously described. Immediately following these, enter the value sets for each ragged variable. For each ragged variable the number of values entered must agree with the value set sizes in NTAB(). For example, consider the value sets for x_3 in Table 4. The sizes for these sets are given in NTAB(11:16) as 2,3,4,3,2,3. The six value sets for x_3 are of these respective sizes and are stored in XT(6:22).

If x_i is a ragged variable and LUP(i) = 3, then each of the value sets for x_i must be entered in XT() as just a pair of numbers representing the first value and the increment. No gaps are to be left in the XT() array between these pairs and the data for x_{i+1}, if any.

B.3.c Specification of YT() for a ragged table

Table 1. A 2-D Grid

X1⇒	−1	2	3	8
X2⇓	Y			
20	1	4	7	10
22	2	5	8	11
27	3	6	9	12

NDIM = 2
NTAB() = (4,3, 0, 0,0)
⇒ (4,3, 1000, 1,5)
XT() = (−1,2,3,8, 20,22,27)
YT() = (1,2,3, 4,5,6, 7,8,9, 10,11,12)

Table 2. A 3-D Grid

X1⇒	3		7		8	
X2⇒	20	22	20	22	20	22
X3⇓	Y					
31	1	5	9	13	17	21
33	2	6	10	14	18	22
35	3	7	11	15	19	23
36	4	8	12	16	20	24

NDIM = 3
NTAB() = (3,2,4, 0, 0,0,0)
⇒ (3,2,4, 1000, 1,4,6)
XT() = (3,7,8, 20,22, 31,33,35,36)
YT() = (1,2,3,4, 5,6,7,8, 9,10,11,12, 13,14,15,16, 17,18,19,20, 21,22,23,24)

Table 3. A 2-D Ragged Table with X2 depending on X1

X1⇒	−1		2		5		8	
	X2	Y	X2	Y	X2	Y	X2	Y
	20	1	21	4	20	6	21	9
	22	2	28	5	24	7	27	10
	27	3			28	8		

NDIM = 2
NTAB() = (4,−1, 0, 0,0, 0, 0,3,2,3,2,−1)
⇒ (4,−1, 1, 1,5, 7, 0,3,5,8,10,−1)
XT() = (−1,2,5,8, 20,22,27, 21,28, 20,24,28, 21,27)
YT() = (1,2,3, 4,5, 6,7,8, 9,10)

The y values must be listed in YT() in lexicographic order of the indexes of the independent variables. For example, if NDIM = 3, YT(1) must be the value of y associated with the first value of x_1, the first value of x_2 allowed to occur with this value of x_1, and the first value of x_3 allowed to occur with these values of x_1 and x_2. If there are more values of x_3 allowed to occur with these values of x_1 and x_2 then the values of y associated with these combinations of values would come next in YT(). The setting of the YT() array is illustrated in Tables 1 – 6. In these tables both the initial and the final contents of NTAB() are given.

B.4 Getting Output of Tables

As verification that things have been set up correctly it can be helpful with complicated ragged tables to see how things are laid out. This data can be seen with a call vary similar to that for SILUPM. Just prior to the call to SILUPM,

CALL SILUPMD(NDIM, X, Y, NTAB, XT, YT, NDEG, LUP, IOPT, EOPT)

This can result in a lot of output, so if possible it may work best for you if you reduce the size of the tables while keeping the logical structure intact.

Table 4. A 3-D Ragged Table with X3 depending on X1 and X2

X1⇒	3				7				8			
X2⇒	20		22		20		22		20		22	
	X3	Y	X3	Y	X3	Y	X3	Y	X3	Y	X3	Y
	30	1	31	3	31	6	30	10	32	13	30	15
	39	2	35	4	33	7	35	11	39	14	36	16
			38	5	36	8	39	12			38	17
					38	9						

NDIM = 3
NTAB() = (3,2,−2, 0, 0,0,0, 0,0, 0,2,3, 4,3, 2,3, −1)
⇒ (3,2,−2, 3, 1,4,6, 10,10, 0,2,5, 9,12, 14,17, −1)
XT() = (3,7,8, 20,22, 30,39, 31,35,38, 31,33,36,38, 30,35,39, 32,39, 30,36,38)
YT() = (1,2, 3,4,5, 6,7,8,9, 10,11,12, 13,14, 15,16,17)

Table 5. A 3-D Ragged Table with X2 depending on X1, and X3 depending on X1 and X2

X1⇒	3						7				8			
X2⇒	20		25		29		21		29		20		28	
	X3	Y	X3	Y	X3	Y	X3	Y	X3	Y	X3	Y	X3	Y
	31	1	30	3	32	6	30	8	31	11	31	13	30	15
	39	2	35	4	37	7	34	9	37	12	38	14	36	16
			38	5			38	10					39	17

NDIM = 3
NTAB() = (3,−1,−2, 0, 0,0,0, 0,0, 0,3,2,2, 0,2,3,2, 3,2, 2,3, −1)
⇒ (3,−1,−2, 2, 1,4,11, 10,14, 0,3,5,7, 0,2,5,7, 10,12, 14,17, −1)
XT() = (3,7,8, 20,25,29, 21,29, 20,28, 31,39, 30,35,38, 32,37, 30,34,38, 31,37, 31,38, 30,36,39)
YT() = (1,2, 3,4,5, 6,7, 8,9,10, 11,12, 13,14, 15,16,17)

Table 6. A 3-D Ragged Table with X3 depending only on X1

X1⇒	3			7			8		
X2⇒		20	22		20	22		20	22
	X3	Y		X3	Y		X3	Y	
	30	1	3	31	5	8	30	11	13
	38	2	4	35	6	9	39	12	14
				39	7	10			

NDIM = 3
NTAB() = (3,2,−1, 0, 0,0,0, 0,0, 0,2,3,2, −1)
⇒ (3,2,−1, 1, 1,4,6, 10,10, 0,2,5,7, −1)
XT() = (3,7,8, 20,22, 30,38, 31,35,39, 30,39)
YT() = (1,2, 3,4, 5,6,7, 8,9,10, 11,12, 13,14)

B.5 Modifications for Double Precision

Change SILUPM to DILUPM, SILUPMD to DILUPMD and the REAL type statement to DOUBLE PRECISION.

C. Examples and Remarks

DRSILUPM below is a sample program that does interpolation in a table of $\sin(x_1x_2)$ given with a spacing of 0.08 in x_1 and a spacing of 0.12 in x_2, degree 8 in x_1 and degree 10 in x_2, and obtains an error estimate. ODSILUPM below gives the results of running DRSILUPM on an IBM PC.

Section C of Chapter 12.1 applies here also. In particular a discontinuity occurring at $x_i = d_i$ can be indicated by having successive XT() values associated with $x_i = d_i$.

D. Functional Description

Interpolations are done by doing nested one dimensional interpolations in the coordinate directions. First the XT() values to be used depending on the value of x_1 are determined. Then the values of XT() for the value of x_2 are determined, etc. For ragged tables, the values of some of the XT() that are to be used depend on indexes selected for lower indexed XT()'s. After the base pointer to XT() and YT() is determined for the last dimension, NDIM, a one dimensional interpolation is done in dimension NDIM. This provides one function value for the next lower dimension, and other values are obtained in a similar way by varying the index in the next to the last dimension. This process gives function values that can be interpolated to provide a function value for the next lower dimension, etc.

Let n_j = NTAB(j), j = 1, 2, ..., NDIM. The n_j must satisfy:

If $n_k > 0$, then $n_j > 0$ for $j < k$, and n_k gives the number of data points in dimension k.

If $n_k < 0$, then information for the current dimension depends on dimensions with indexes from 1 to $|n_k|$. If $k \neq$ NDIM then $n_k = -k + 1$, else $1 \leq |n_k| <$ NDIM.

Tables with an $n_k < 0$ are called ragged, others are said to have data defined on a grid. When data are defined on a grid, no extra information is required. In the ragged case, the number of XT() values and their values depend on the indexes from dimensions with a smaller index. The user must provide this information. So users with data on a grid don't have to deal with the complexities of the ragged case, we give the formula for finding XT() data although for the case of ragged tables some of the symbols used rely on things defined later. Let

$$\xi_k = \begin{cases} 2 & \text{LUP}_k = 3, \quad n_k > 0 \\ n_k & \text{LUP}_k \neq 3, \quad n_k > 0 \\ 2\mu_k & \text{LUP}_k = 3, \quad n_k < 0 \\ S_{k,\mu_k} & \text{LUP}_k \neq 3, \quad n_k < 0 \end{cases} \tag{1}$$

then XT() data for interpolation with respect to the k^{th} dimension starts in

$$\text{XT}\left(1 + \eta_k + \sum_{j=1}^{k-1} \xi_j\right), \quad \text{where} \tag{2}$$

$$\eta_k = \begin{cases} 0 & \text{if } n_k > 0 \\ 2\left(p_{-n_k} + i_{-n_k} - 1\right) & \text{if } n_k < 0 \ \&\ \text{LUP}_k = 3 \\ p_{1-n_k} & \text{if } n_k < 0 \ \&\ \text{LUP}_k \neq 3 \end{cases} \tag{3}$$

On the first call to the program, NTAB(NDIM+1) is set to $|n_k|$, where k is the first index for which $n_k < 0$ (or to 1000 if all $n_k > 0$), and NTAB(NDIM+k+1) is set to $1 + \sum_{j=1}^{k-1} \xi_j, k = 1, 2, ...,$ NDIM. User's with data on a grid need not know about μ, p, S or anything else beyond this point.

Ragged tables require information on the nature of the raggedness. This information must be supplied in what we call lexicographic order. The first data supplied are for the r^{th} dimension, where r is the index of the first dimension that is ragged. Let $e = |\text{NTAB}(r)|$, and the indexes selected from dimensions 1 to e be (i_1, i_2, ..., i_e). The data $K_{r,j}$ supplied for this dimension define the number of YT() values defined for all the possible values of the indexes from dimensions 1 to e. $K_{r,0}$ must be 0, and $K_{r,1}$ is the number of YT() values that correspond to having i_1, ..., i_e = 1, $K_{r,2}$ to having i_e = 2, and all of the rest 1, etc. That is a lexicographic ordering of the indexes (i_1, ..., i_e). Following the data for dimension r comes the data for dimension $r + 1$, if any, etc. To enhance error checking, the K's for the last dimension must be followed by -1. To describe how the program keeps track of this information we introduce some additional notation. Let μ_k denote the index of the last j for the $K_{k,j}$. Then

$$\mu_k = \begin{cases} -1 & \text{if } n_k > 0 \\ \prod_{j=1}^{e} n_j & \text{if } k = r \\ S_{k-1,\mu_{k-1}} & \text{if } k > r, \quad \text{where} \end{cases} \tag{4}$$

$$S_{k,j} = \sum_{m=0}^{j} K_{k,m}, \quad j = 0, 1, ..., \mu_k. \tag{5}$$

The $S_{k,j}$ replace the $K_{k,j}$ in storage. Let $L_1 = 3 \times$ NDIM + 1, and $L_k = L_{k-1} + \mu_k + 1$, $k > 1$. Initially NTAB($L_k + j$) contains $K_{k,j}$; these are replaced by the

$S_{k,j}$ on the first call. L_k is stored in NTAB($2 \times$ NDIM$+ k + 1$), k = 1, 2, ..., NDIM $-$ 1. With what has been defined so far, it is now possible to compute the values stored in NTAB(NDIM+k+1) as described just below Eq. (3).

The notation introduced below defines the rest of the information required to find the XT() values as well as the values of YT() to be accessed in the one dimensional interpolations. In both cases the data are in lexicographic order as defined above and this is all the user need know in order to save the required information. As above, let n_r be the first $n_k < 0$, $e = -n_r$, and define p_k to be a pointer for accessing information that depends on the indexes i_1, i_2, ..., i_{k-1}. Recall that if $r \neq$ NDIM, then $e = r - 1$. With

$$\begin{aligned} p_1 &= 0 \\ p_{k+1} &= n_{k+1}\,(i_k - 1 + p_k)\,, \quad k = 1, 2, ..., e-1 \\ p_{e+1} &= S_{r,p_e+i_e-1}, \ \text{ and for } k = e+1, \ ..., \ \text{NDIM}-1, \\ p_{k+1} &= \begin{cases} n_k p_k + (i_k - 1)\,(S_{r,p_e+i_e} - S_{r,p_e+i_e-1})\,, & \\ & r = \text{NDIM}, \\ S_{k+1,p_k+i_k-1}, & r \neq \text{NDIM}, \end{cases} \end{aligned} \tag{6}$$

the value of YT() corresponding to $(i_1, \, i_2, \, ..., \, i_{NDIM})$ is in YT($p_{\text{NDIM}} + i_{\text{NDIM}}$), and YT($p_{\text{NDIM}} + 1$) is passed to the one-dimensional interpolation routine along with the XT() as defined by Eqs. (2), (3) and (6).

To compute the storage required for computing and storing derivatives, let $m_{i,k}$ denote the number of derivatives in dimension i with the sum of derivative orders exactly k, $M_{i,k} = \sum_{j=0}^{k} m_{i,j}$, and t_i = the number of derivatives with respect to x_i. Since we start by computing derivatives in the last dimension, NDIM, $m_{NDIM,k} = 1$ for $k \leq t_{\text{NDIM}}$, and is 0 otherwise. Also, $m_{i,0} \equiv 1$ since there will be exactly one interpolated value. Differentiating the results from a higher dimension 0, 1, ..., min(k, t_i) times gives $m_{i,k} = \sum_{j=max(0,k-t_i)}^{k} m_{i+1,j}$, and with the convention that $M_{i,k} = 0$ for $k < 0$,

$$M_{i,k} = M_{i,k-1} + M_{i+1,k} - M_{i+1,k-t_i-1} \tag{7}$$

Let $\ell_i = \min\{\sum_{j=i}^{NDIM} t_j, \text{ L3 (from options)}\}$. Then the desired recursion for computing the $M_{i,k}$ is given by

$$\begin{aligned} M_{i,0} &= 1 \\ M_{i,k} &= M_{i,k-1} + M_{i+1,k} - M_{i+1,k-t_i-1}, \ k = 1, 2, ..., \ell_i. \end{aligned} \tag{8}$$

The D_i referred to in the description of option 3 are given by $D_i = M_{i,\ell_i}$.

E. Error Procedures and Restrictions

Values of IOPT(1) < 0 ordinarily cause an error message to be printed, and those < -2 do not ordinarily result in a return to the user. One can change the action on errors by calling the message/error routine MESS of Chapter 19.3 before calling this routine.

F. Supporting Information

The source language is ANSI Fortran 77.

Entry	Required Files
DILUPM	AMACH, DILUP, DILUPM, DILUPMD, DMESS, MESS, OPTCHK
SILUPM	AMACH, MESS, OPTCHK, SILUP, SILUPM, SILUPMD, SMESS

Subroutine designed and written by Fred T. Krogh, JPL, May 1991. SILUPMD added, Fred T, Krogh Math à la Carte, May 2006.

DRSILUPM

```
      program drsilm
c>> 2001-05-22 DRSILM  Krogh Minor change for making .f90 version.
c>> 1994-10-19 DRSILM  Krogh  Changes to use M77CON
c>> 1992-03-04 DRSILM  Krogh  Initial Code.
c--S replaces "?": DR?ILM, ?ILUPM
c Demonstration driver for SILUPM.
c Given a table of sin(xy), for x = 0, .08, .16, ..., and y = 0, .12, .2
c ... interpolates for (x,y) = (2.7, -.1), (.93, .05), (.765, .87) and
c obtains an error estimate.
c
      integer ND1, ND2, NDEG1, NDEG2, NUMTAB, NDIM, NX, NDIMT, NEOPT
      parameter (ND1=50, ND2=40, NDEG1=8, NDEG2=10, NUMTAB=ND1*ND2,
     1  NDIM=2, NX=3, NDIMT=2*NDIM+1, NEOPT=2*(2*NDIM+NDEG1)+1)
      integer  I, J, IOPT(4), NTAB(NDIMT), NDEG(NDIM), LUP(NDIM)
      real             X1(NX), X2(NX), YT(NUMTAB), YT2(ND2, ND1),
     1   XT(2*NDIM), X(NDIM), Y, EOPT(NEOPT), H1, H2, ANS
      parameter (H1 = .08E0, H2 = .12E0)
      equivalence (YT, YT2)
      data X1 / 2.7E0, .93E0, .765E0 /
      data X2 / -.1E0, .05E0, .87E0 /
c                        Set IOPT to get an error estimate.
      data IOPT / 0, NEOPT, 1, 0 /
      data NTAB(1), NTAB(2), NTAB(3) / ND1, ND2, 0 /
      data NDEG, LUP / NDEG1, NDEG2, 3, 3 /
c
c Compute the XT and YT tables
      XT(1) = 0.E0
      XT(2) = H1
      XT(3) = 0.E0
      XT(4) = H2
      do 20 I = 1, ND1
         do 10 J = 1, ND2
            YT2(J, I) = sin(real((I-1)*(J-1)) * H1 * H2)
   10    continue
   20 continue
      print *,
     1  'IOP(1)    X1      X2          Y      Est. Error True Error'
      do 50 I = 1, NX
         X(1) = X1(I)
         X(2) = X2(I)
         ANS = sin(X(1)*X(2))
            call SILUPM (NDIM,X,Y,NTAB,XT,YT,NDEG,LUP,IOPT,EOPT)
            print '(I5, 2F9.4, F12.8, 1P,E10.2, E11.2)',
     1        IOPT(1), X1(I), X2(I), Y, EOPT(1), Y - ANS
   50 continue
      stop
      end
```

ODSILUPM

```
IOP(1)    X1        X2          Y         Est. Error  True Error
    1   2.7000   -0.1000  -0.26674628   1.22E-04   -1.48E-05
    0   0.9300    0.0500   0.04648293   3.95E-07   -3.13E-07
    0   0.7650    0.8700   0.61749178   1.80E-07   -5.96E-08
```