6.3 Basic Linear Algebra Subprograms (BLAS1)

A. Purpose

This is a set of subroutine and function subprograms for basic mathematical operations on a single vector or a pair of vectors. The operations provided are those commonly used in algorithms for numerical linear algebra problems, *e.g.*, problems involving systems of equations, least-squares, matrix eigenvalues, optimization, etc.

B. Usage

Described below under B.1 through B.13 are:

B.1 Vector Arguments

For subprograms of this package a vector is specified by three arguments, say N, SX, and INCX, where

N denotes the number of elements in the vector,

SX (or DX or CX) identifies the array containing the vector, and

INCX is the (signed) storage increment between successive elements of the vector.

Let x_i, $i = 1$, ..., N, denote the vector stored in the array SX(). Within a subprogram of this package the array argument, SX, will be declared as

REAL SX($*$)

In the common case of INCX $= 1$, x_i is stored in SX(i). More generally, if INCX ≥ 0, x_i is stored in SX($1 + (i - 1) \times$ INCX), and if INCX < 0, x_i is stored in X($1 + (\mathrm{N} - i) \times |\mathrm{INCX}|$).

With indexing as specified above, looping operations within the subprograms are performed in the order i=1, 2, ..., N.

Only positive values of INCX are allowed for subprograms that have a single vector argument. For subprograms having two vector arguments, the two increment parameters, INCX and INCY, may independently be positive, zero, or negative.

Due to the Fortran 77 rules for argument association and array storage, the actual argument playing the role of SX is not limited to being a singly dimensioned array. It may be an array of any number of dimensions, or it may be an array element with any number of subscripts. The standard does not permit this argument to be a simple variable, however.

If the actual argument is an array element, *i.e.*, a subscripted array name, then it identifies x_1, the first element of the vector, if INCX ≥ 0, and x_N, the last element of the vector, if INCX < 0.

The most common reason for using an increment value different from one is to operate on a row of a matrix. For example, if an array is declared as

DOUBLE PRECISION A(5, 10)

The third row of A(,) begins at element A(3,1) and has a storage spacing of 5 (double precision) storage locations between successive elements. Thus this row-vector of length 10 would be described by the parameter triple (N, DX, INCX) = (10, A(3,1), 5) in calls to subprograms of this package.

See Section C for examples illustrating the specifications of vector arguments.

In the following subprogram descriptions we assume the following declarations have been made for any subprograms that use these variable names

INTEGER N, INCX, INCY

REAL SX(mx), **SY**(my)

DOUBLE PRECISION DX(mx), **DY**(my)

COMPLEX CX(mx), **CY**(my)

In the common case of INCX $= 1$, mx must satisfy $mx \geq$ N. More generally, mx must satisfy $mx \geq 1 + (\mathrm{N} - 1) \times |\mathrm{INCX}|$. Similarly, my must be consistent with N and INCY.

In Sections B.2 through B.13 we use the convention that **x** denotes the vector contained in the storage array SX(), DX(), or CX(), a denotes the scalar value contained in SA, DA, or CA, etc.

B.2 Dot Product Subprograms

REAL SDOT, SDSDOT, SB, SW

DOUBLE PRECISION DDOT, DSDOT, DW

COMPLEX CDOTC, CDOTU, CW

The first four subprograms each compute

$$w = \sum_{i=1}^{N} x_i y_i$$

Single precision:

SW = SDOT(N, SX, INCX, SY, INCY)

Double precision:

DW = DDOT(N, DX, INCX, DY, INCY)

Single precision data. Uses double precision arithmetic internally and returns a double precision result:

DW = DSDOT(N, SX, INCX, SY, INCY)

Complex (unconjugated):

CW = CDOTU(N, CX, INCX, CY, INCY)

CDOTC computes

$$w = \sum_{i=1}^{N} \bar{x}_i y_i$$

where $\bar{x}_i$ denotes the complex conjugate of the given x_i. This is the usual inner product of complex N-space.

CW = CDOTC(N, CX, INCX, CY, INCY)

SDSDOT computes

$$w = b + \sum_{i=1}^{N} x_i y_i$$

using single precision data, double precision internal arithmetic, and converting the final result to single precision:

SW = SDSDOT(N, SB, SX, INCX, SY, INCY)

In each of the above six subprograms the value of the summation from 1 to N will be set to zero if $N \le 0$.

B.3 Scalar Times a Vector Plus a Vector

REAL SA

DOUBLE PRECISION DA

COMPLEX CA

Given a scalar, a, and vectors, $\mathbf{x}$ and $\mathbf{y}$, each of these subroutines replaces $\mathbf{y}$ by $a\mathbf{x} + \mathbf{y}$. If $a = 0$ or $N \le 0$, each subroutine returns, doing no computation.

Single precision:

CALL SAXPY (N, SA, SX, INCX, SY, INCY)

Double precision:

CALL DAXPY (N, DA, DX, INCX, DY, INCY)

Complex:

CALL CAXPY (N, CA, CX, INCX, CY, INCY)

B.4 Construct a Givens Plane Rotation

REAL SA, SB, SC, SS

DOUBLE PRECISION DA, DB, DC, DS

Single precision:

CALL SROTG (SA, SB, SC, SS)

Double precision:

CALL DROTG (DA, DB, DC, DS)

Given a and b, each of these subroutines computes c and s, satisfying

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

subject to $c^2 + s^2 = 1$ and $r^2 = a^2 + b^2$. Thus the matrix involving c and s is an orthogonal (rotation) matrix that transforms the second component of the vector $[a, b]^t$ to zero. This matrix is used in certain least-squares and eigenvalue algorithms.

If $r = 0$ the subroutine sets $c = 1$ and $s = 0$. Otherwise the sign of r is set so that $sgn(r) = sgn(a)$ if $|a| > |b|$, and $sgn(r) = sgn(b)$ if $|a| \le |b|$. Then, $c = a/r$ and $s = b/r$.

Besides setting c and s, the subroutine stores r in place of a and another number, z, in place of b. The number z is rarely needed. See [1]– [4] in Section D for a description of z.

These subroutines are designed to avoid extraneous overflow or underflow in cases where r^2 is outside the exponent range of the computer arithmetic but r is within the range.

B.5 Apply a Plane Rotation

REAL SC, SS

DOUBLE PRECISION DC, DS

Single precision:

CALL SROT (N, SX, INCX, SY, INCY, SC, SS)

Double precision:

CALL DROT (N, DX, INCX, DY, INCY, DC, DS)

Given vectors, **x** and **y**, and scalars, c and s, this subroutine replaces the 2×N matrix

$$\begin{bmatrix} \mathbf{x}^t \\ \mathbf{y}^t \end{bmatrix} \quad \text{by the } 2 \times \text{N matrix} \quad \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}^t \\ \mathbf{y}^t \end{bmatrix}.$$

If $\text{N} \leq 0$ or if $c = 1$ and $s = 0$, these subroutines return, doing no computation.

B.6 Construct a Modified Givens Transformation

REAL SD1, SD2, SX1, SX2, SPARAM(5)

DOUBLE PRECISION DD1, DD2, DX1, DX2, DPARAM(5)

Single precision:

CALL SROTMG (SD1, SD2, SX1, SX2, SPARAM)

Double precision:

CALL DROTMG (DD1, DD2, DX1, DX2, DPARAM)

The input quantities d_1, d_2, x_1, and x_2, define a 2-vector $[w_1\ w_2]^t$ in partitioned form as

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} d_1^{1/2} & 0 \\ 0 & d_2^{1/2} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The subroutine determines the modified Givens rotation matrix H that transforms x_2 and thus w_2 to zero. It also replaces d_1, d_2, and x_1 with δ_1, δ_2, and ξ_1, respectively. These quantities satisfy

$$\begin{bmatrix} \omega \\ 0 \end{bmatrix} = \begin{bmatrix} \delta_1^{1/2} & 0 \\ 0 & \delta_2^{1/2} \end{bmatrix} \cdot H \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \delta_1^{1/2} & 0 \\ 0 & \delta_2^{1/2} \end{bmatrix} \cdot \begin{bmatrix} \xi_1 \\ 0 \end{bmatrix},$$

with $\omega = \pm(w_1^2 + w_2^2)^{1/2}$. A representation of the matrix H will be stored by this subroutine into SPARAM() or DPARAM() for subsequent use by subroutines SROTM or DROTM.

See the Appendix in [1] for more details on the computation and storage of H.

Most of the time the matrix H will be constructed to have two elements equal to +1 or −1. Thus multiplication of a 2-vector by H can be programmed to be faster than multiplication by a general 2×2 matrix. This is the motivation for using a modified Givens matrix rather than a standard Givens matrix; however, these matrix multiplications must represent a very significant percentage of the execution time of an application program in order for the extra complexity of using the modified Givens matrix to be worthwhile.

B.7 Apply a Modified Givens Transformation

REAL SPARAM(5)

DOUBLE PRECISION DPARAM(5)

Single precision:

CALL SROTM (N, SX, INCX, SY, INCY, SPARAM)

Double precision:

CALL DROTM (N, DX, INCX, DY, INCY, DPARAM)

Given vectors, **x** and **y**, and a representation of an H matrix constructed by SROTMG or DROTMG in SPARAM() or DPARAM(), this subroutine replaces the 2×N matrix

$$\begin{bmatrix} \mathbf{x}^t \\ \mathbf{y}^t \end{bmatrix} \quad \text{by} \quad H \cdot \begin{bmatrix} \mathbf{x}^t \\ \mathbf{y}^t \end{bmatrix}.$$

Due to the special form of the matrix, H, this matrix multiplication generally requires only 2N multiplications and 2N additions rather than the 4N multiplications and 2N additions that would be required if H were an arbitrary 2×2 matrix.

If $\text{N} \leq 0$ or H is the identity matrix, these subroutines return immediately.

B.8 Copy a Vector x to y

Single precision:

CALL SCOPY (N, SX, INCX, SY, INCY)

Double precision:

CALL DCOPY (N, DX, INCX, DY, INCY)

Complex:

CALL CCOPY (N, CX, INCX, CY, INCY)

Each of these subroutines copies the vector $\mathbf{x}$ to $\mathbf{y}$. If N ≤ 0 the subroutine returns immediately.

B.9 Swap Vectors x and y

Single precision:

CALL SSWAP (N, SX, INCX, SY, INCY)

Double precision:

CALL DSWAP (N, DX, INCX, DY, INCY)

Complex:

CALL CSWAP (N, CX, INCX, CY, INCY)

This subroutine interchanges the vectors $\mathbf{x}$ and $\mathbf{y}$. If N ≤ 0, the subroutine returns immediately.

B.10 Euclidean Norm of a Vector

REAL SNRM2, SCNRM2, SW

DOUBLE PRECISION DNRM2, DW

Single precision:

SW = SNRM2(N, SX, INCX)

Double precision:

DW = DNRM2(N, DX, INCX)

Complex data, REAL result:

SW = SCNRM2(N, CX, INCX)

Each of these subprograms computes

$$w = \left[\sum_{i=1}^{N} |x_i|^2\right]^{1/2}.$$

If N ≤ 0, the result is set to zero.

These subprograms are designed to avoid overflow or underflow in cases in which w^2 is outside the exponent range of the computer arithmetic but w is within the range.

B.11 Sum of Magnitude of Vector Components

REAL SASUM, SCASUM, SW

DOUBLE PRECISION DASUM, DW

SASUM and DASUM compute

$$w = \sum_{i=1}^{N} |x_i|.$$

Single precision:

SW = SASUM(N, SX, INCX)

Double precision:

DW = DASUM(N, DX, INCX)

Given a complex vector, $\mathbf{x}$, SCASUM computes a REAL result for the expression:

$$w = \sum_{i=1}^{N} |\Re x_i| + |\Im x_i|.$$

SW = SCASUM(N, CX, INCX)

If N ≤ 0 these subprograms set the result to zero.

B.12 Vector Scaling

REAL SA

DOUBLE PRECISION DA

COMPLEX CA

Given a scalar, a, and vector, $\mathbf{x}$, each of these subroutines replaces the vector, $\mathbf{x}$, by the product $a\mathbf{x}$.

Single precision:

CALL SSCAL (N, SA, SX, INCX)

Double precision:

CALL DSCAL (N, DA, DX, INCX)

Complex:

CALL CSCAL (N, CA, CX, INCX)

Given REAL a and complex $\mathbf{x}$, the subroutine CSSCAL replaces $\mathbf{x}$ by the product $a\mathbf{x}$.

CALL CSSCAL (N, SA, CX, INCX)

If N ≤ 0 these subroutines return immediately.

B.13 Find Vector Component of Largest Magnitude

INTEGER ISAMAX, IDAMAX, ICAMAX, IMAX

ISAMAX and IDAMAX each determine the smallest i such that

$$|x_i| = \max\{|x_j| : j = 1, ..., N\}$$

REAL vector, integer result:

$$\boxed{\textbf{IMAX = ISAMAX(N, SX, INCX)}}$$

Double precision vector, integer result:

$$\boxed{\textbf{IMAX = IDAMAX(N, DX, INCX)}}$$

Given a complex vector, $\mathbf{x}$, ICAMAX determines the smallest i such that

$$|\Re x_i| + |\Im x_i| = \max\{|\Re x_j| + |\Im x_j| : j = 1, ..., N\}$$

$$\boxed{\textbf{IMAX = ICAMAX(N, CX, INCX)}}$$

If N $\leq$ 0, each of these subprograms sets the integer result to zero.

C. Examples and Remarks

The program, DRDBLAS1, and its output, ODDBLAS1, illustrate the use of various BLAS1 subprograms to compute matrix-vector and matrix-matrix products.

Problems (1) and (2) show two ways of computing the matrix-vector product, $A\mathbf{b}$. Using DDOT accesses the elements of A by rows. Using DAXPY accesses A by columns. The column ordering has an efficiency advantage on virtual memory systems since Fortran stores arrays by columns and there will therefore be fewer page faults.

Problem (3) illustrates multiplication by the transpose of a matrix without transposing the matrix in storage.

Problem (4) illustrates matrix-matrix multiplication. This operation could also be programmed to use DAXPY rather than DDOT.

These examples also illustrate the feature that matrix dimensions and storage array dimensions need not be the same. Thus, in DRDBLAS1, a 2×3 matrix A is stored in a 5×10 storage array A(,).

The program, DRDBLAS2, with output, ODDBLAS2, illustrates a complete algorithm for solving a linear least-squares problem. The algorithm first performs sequential accumulation of data into a triangular matrix, using DROTG and DROT to build and apply Givens orthogonal transformations. It then solves the triangular system using DCOPY and DAXPY. This is a very reliable algorithm.

The problem solved is the determination of coefficients c_1, c_2, and c_3 in the expression $c_1 + c_2 x + c_3 \exp(-x)$ to produce a least-squares fit to the data given in XTAB() and YTAB(). Note that by changing dimension parameters, and the statements assigning values to the array W(), DRDBLAS2 could be altered to solve any specific linear least-squares problem.

D. Functional Description

This set of subprograms is described in more detail in [1]. For discussion of the standard and modified Givens orthogonal transformations and their use in least-squares computations see [5].

There is an error in [1] in the discussion of the parameter, z, which is returned by SROTG or DROTG. This error was corrected by [3], after which [4] corrected an error in [3].

References

1. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for Fortran usage*, **ACM Trans. on Math. Software 5**, 3 (Sept. 1979) 308–323.
2. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Algorithm 539: Basic Linear Algebra Subprograms for Fortran usage [F1]*, **ACM Trans. on Math. Software 5**, 3 (Sept. 1979) 324–325.
3. David S. Dodson and Roger G. Grimes, *Remark on "Algorithm 539: Basic Linear Algebra Subprograms for Fortran usage [F1]"*, **ACM Trans. on Math. Software 8**, 4 (Dec. 1982) 403–404.
4. David S. Dodson, *Corrigendum: Remark on "Algorithm 539: Basic Linear Algebra Subroutines for FORTRAN usage"*, **ACM Trans. on Math. Software 9**, 1 (March 1983) 140–140.
5. Charles L. Lawson and Richard J. Hanson, **Solving Least-Squares Problems**, Prentice-Hall, Englewood Cliffs, N. J. (1974) 340 pages.
6. Fred T. Krogh, **On the Use of Assembly Code for Heavily Used Modules in Linear Algebra**. Internal Technical Memorandum 303, Jet Propulsion Laboratory, Pasadena, CA (May 1972).
7. R. Hanson, F. Krogh, and C. Lawson, **A proposal for standard linear algebra subprograms**. Internal Technical Memorandum 33–660, Jet Propulsion Laboratory (Nov. 1973).
8. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson, *An extended set of FORTRAN Basic Linear Algebra Subprograms*, **ACM Trans. on Math. Software 14**, 1 (March 1988) 1–17.

9. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff, *A set of level 3 Basic Linear Algebra Subprograms*, **ACM Trans. on Math. Software 16**, 1 (March 1990) 1–17.

10. R. J. Hanson and F. T. Krogh, *Algorithm 653: Translation of Algorithm 539: PC-BLAS Basic Linear Algebra Subprograms for FORTRAN usage with the INTEL 8087, 80287 numeric data processor*, **ACM Trans. on Math. Software 13**, 3 (Sept. 1987) 311–317.

E. Error Procedures and Restrictions

These subprograms do not issue any error messages. If INCX ≤ 0 in any of the subprograms having a single vector argument the results are unpredictable.

A value of $N = 0$ is a valid special case. Values of $N < 0$ are treated like $N = 0$.

F. Supporting Information

The source language is ANSI Fortran 77.

Each program unit has a single entry with the same name as the program unit. No program units contain external references.

As a result of experiments done in [6], a BLAS-type package was originally proposed in [7]. The package was subsequently developed and tested over the period 1973–77 as an ACM SIGNUM committee project, with the final product being announced in [1]. This package was subsequently used as a component in many other numerical software packages and optimized machine-language versions were produced for a number of different computer systems.

This package was identified as BLAS when it appeared in 1979 [1], however it is now identified as "BLAS1", or "Level 1 BLAS" since publication of BLAS2 [8] for matrix-vector operations and BLAS3 [9] for matrix-matrix operations. These latter two packages support more efficient use of vector registers, processor cache, and parallel processors than is possible in BLAS1.

Adapted to Fortran 77, by C. Lawson and S. Chiu, JPL, January 1984. Replaced L2 norm routines with new versions which avoid undeflow in all cases and which don't use assigned go to's, F. Krogh, May 1998.

All entries need one file of the same name, except for DNRM2, SCNRM2, and SNRM2 all of which also require AMACH.

Entries

CAXPY	**CCOPY**	**CDOTC**	**CDOTU**
CSCAL	**CSSCAL**	**CSWAP**	**DASUM**
DAXPY	**DCOPY**	**DDOT**	**DNRM2**
DROT	**DROTG**	**DROTM**	**DROTMG**
DSCAL	**DSDOT**	**DSWAP**	**ICAMAX**
IDAMAX	**ISAMAX**	**SASUM**	**SAXPY**
SCASUM	**SCNRM2**	**SCOPY**	**SDOT**
SDSDOT	**SNRM2**	**SROT**	**SROTG**
SROTM	**SROTMG**	**SSCAL**	**SSWAP**

DRDBLAS1

```
c      program DRDBLAS1
c>> 1996-06-18 DRDBLAS1 Krogh  Minor format change for C conversion.
c>> 1996-05-28 DRDBLAS1 Krogh Added external statement.
c>> 1994-10-19 DRDBLAS1 Krogh  Changes to use M77CON
c>> 1992-03-24 DRDBLAS1 CLL Removed reference to DBLE() function.
c>> 1991-12-02 DRDBLAS1 CLL
c>> 1991-07-25 DRDBLAS1 CLL
c>> 1987-12-09 DRBLAS1  Lawson  Initial Code.
c
c      Demonstrate usage of DAXPY, DCOPY, and DDOT from the BLAS
c      by computing
c      (1) p = A * b                 using DDOT
c      (2) q = A * b                 using DCOPY & DAXPY
c      (3) r = (A Transposed) * p    using DDOT
c      (4) S = A * E                 using DDOT
c      ------------------------------------------------------------------
c--D replaces "?": DR?BLAS1, ?AXPY, ?COPY, ?DOT
c      ------------------------------------------------------------------
      external DDOT
      double precision DDOT
      integer M2, M3, M4, N2, N3, N4
      parameter ( M2=5, M3=10, M4=12 )
```

```
      parameter ( N2=2, N3=3, N4=4 )
      integer I, J
      double precision A(M2,M3), E(M3,M4), S(M2,M4)
      double precision B(M3), P(M2), Q(M2), R(M3)
      double precision ZERO(1)
c
      data ZERO(1) / 0.0d0 /
      data (A(1,J),J=1,N3) /  2.0d0, -4.0d0, 3.0d0 /
      data (A(2,J),J=1,N3) / -5.0d0, -2.0d0, 6.0d0 /
      data (B(J),J=1,N3)   /  7.0d0, -3.0d0, 5.0d0 /
      data (E(1,J),J=1,N4) / -4.0d0, 2.0d0,  3.0d0, -6.0d0 /
      data (E(2,J),J=1,N4) /  7.0d0, 5.0d0, -6.0d0, -3.0d0 /
      data (E(3,J),J=1,N4) /  3.0d0, 4.0d0, -2.0d0,  5.0d0 /
c     ------------------------------------------------------------------
c
c                                  1. p = A * b using DDOT
c
      do 10 I = 1, N2
        P(I) = DDOT(N3,A(I,1),M2,B,1)
   10 continue
c
c                                  2. q = A * b using DCOPY and DAXPY
c
      call DCOPY(N2,ZERO,0,Q,1)
      do 20 J = 1, N3
        call DAXPY(N2,B(J),A(1,J),1,Q,1)
   20 continue
c
c                                  3. r = (A Transposed) * p using DDOT
c
      do 30 J = 1, N3
        R(J) = DDOT(N2,A(1,J),1,P,1)
   30 continue
c
c                                  4. S = A * E using DDOT
c
      do 50 I = 1, N2
        do 40 J = 1, N4
          S(I,J) = DDOT(N3,A(I,1),M2,E(1,J),1)
   40   continue
   50 continue
c
      print*,'DRDBLAS1..  Demo driver for DAXPY, DCOPY, and DDOT'
      print'(/'' P() = '', 7x,4f8.1)', (P(J),J=1,N2)
      print'(/'' Q() = '',7x,4f8.1)', (Q(J),J=1,N2)
      print'(/'' R() = '',7x,4f8.1)', (R(J),J=1,N3)
      print'(/'' S(,) = '')'
      do 60 I = 1,N2
         print'(''   Row '',i2,5x,4f8.1)', I,(S(I,J),J=1,N4)
   60 continue
      stop
      end
```

ODDBLAS1

```
DRDBLAS1..  Demo driver for DAXPY, DCOPY, and DDOT

P() =           41.0      1.0

Q() =           41.0      1.0

R() =           77.0   -166.0    129.0

S(,) =
  Row  1       -27.0     -4.0     24.0     15.0
  Row  2        24.0      4.0    -15.0     66.0
```

DRDBLAS2

```
c     program DRDBLAS2
c>> 1996-06-18 DRDBLAS2 Krogh  Minor format change for C conversion.
c>> 1994-10-19 DRDBLAS2 Krogh  Changes to use M77CON
c>> 1991-11-27 DRDBLAS2 CLL
c>> 1987-12-09 Lawson  Initial Code.
c
c     Demonstrates the use of BLAS subroutines DROTG, DROT, DAXPY,
c     and DCOPY to implement an algorithm for solving a linear
c     least squares problem using sequential accumulation of the
c     data and Givens orthogonal transformations.
c     YTAB() contains rounded values of -2 + 2*X + 3*Exp(-X)
c     ------------------------------------------------------------------
c--D replaces "?": DR?BLAS2, ?ROTG, ?ROT, ?AXPY, ?COPY
c     ------------------------------------------------------------------
      integer MC, MC1, MXY
      parameter ( MC=3, MC1=MC+1, MXY=11 )
      integer IXY, J, NC, NC1, NXY
      double precision X, XTAB(MXY), Y, YTAB(MXY), W(MC1)
      double precision C, RG(MC1,MC1), S
      double precision COEF(MC), DIV, ESTSD, ZERO(1)
c
      data XTAB / 0.0d0,  .1d0,  .2d0,    .3d0,   .4d0,  .5d0,
     *             .6d0,  .7d0,  .8d0,    .9d0,  1.0d0 /
      data YTAB / 1.00d0, .91d0, .86d0,   .82d0,  .81d0, .82d0,
     *             .85d0, .89d0, .95d0,  1.02d0, 1.10d0 /
      data NXY, NC / MXY, MC /
      data ZERO(1) / 0.0d0 /
c     ------------------------------------------------------------------
      NC1 = NC + 1
      call DCOPY(MC1*MC1, ZERO, 0, RG, 1)
      do 20 IXY = 1, NXY
        X = XTAB(IXY)
        Y = YTAB(IXY)
c                                     Build new row of [A:B] in W().
        W(1) = 1.0d0
        W(2) = X
        W(3) = exp(-X)
        W(4) = Y
c                                     Process W() into [R:G].
```

```
      do 10 J = 1, NC
        call DROTG(RG(J,J),W(J),C,S)
        call DROT(NC1-J,RG(J,J+1),MC1,W(J+1),1,C,S)
10    continue
      call DROTG(RG(NC1,NC1),W(NC1),C,S)
20 continue
c                                   Begin: Solve triangular system.
      call DCOPY(NC,RG(1,NC1),1,COEF,1)
      do 30 J = NC, 1, -1
        DIV = RG(J,J)
        if (DIV .eq. 0.0d0) then
          print '(''ERROR:ZERO DIVISOR AT J ='', I2)', J
          stop
        end if
        COEF(J) = COEF(J) / DIV
        call DAXPY(J-1,-COEF(J),RG(1,J),1,COEF,1)
30    continue
c                                   End: Solve triangular system.
c
     print '('' Solution: COEF() = '',3f8.3)',(COEF(J),J=1,NC)
     ESTSD = abs(RG(NC1,NC1)) / sqrt(DBLE(NXY-NC))
     print '(/'' Estimated Std. Dev. of data errors ='',f9.5)', ESTSD
     stop
     end
```

ODDBLAS2

```
Solution: COEF() =   -1.968   1.979   2.966

Estimated Std. Dev. of data errors =  0.00279
```