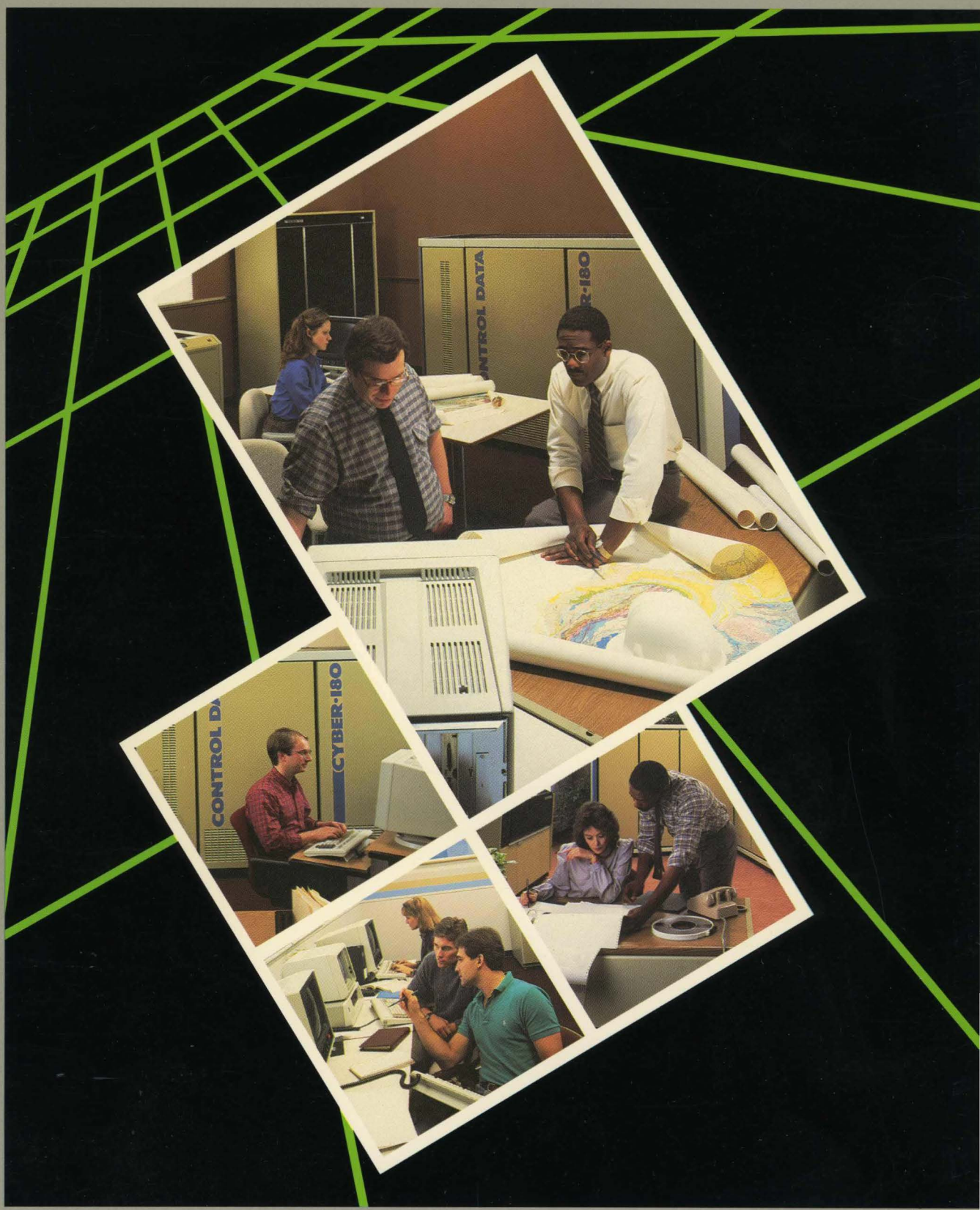
FORTRAN for NOS/VE
LIB99

CONTROL DATA



Usage

60485915

FORTRAN for NOS/VE
LIB99

Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60485915

Manual History

Revision	System Version/ PSR Level	Product Level	Date
A	1.2.1/670	1.0	December 1986
B	1.2.2/678	1.0	May 1987
C	1.2.3/688	1.0	September 1987

This revision:

This manual is revision C and reflects LIB99 for NOS/VE at release 1.2.3, PSR level 688. This revision reflects technical and editorial changes as well as the following new feature descriptions: subprograms F64TR4, F64TSR, F64TSRN, F64TVF, I64TSI, I64TI2, I64TWI, I64TI4, MOVB, R4TF64, SIT164, I2T164, SRTF64, SRNTF64, VFTF64, WITI64, and I4T164.

Contents

About This Manual

About This Manual

About This Manual

CONTROL DATA® LIB99 is a library of FORTRAN callable subroutines written for use with NOS/VE on the CYBER 180.

Audience for This Manual

You should be familar with FORTRAN Version 1 for NOS/VE. In addition, you should know how to create and run jobs under NOS/VE.

Conventions

The following conventions are used in this manual:

UPPERCASE	Subroutine and function names and parameters appear in uppercase.
numbers	All numbers are base 10 unless otherwise noted.

Submitting Comments

Please use the comment sheet in the back of this manual for submitting comments regarding this manual.

Introduction 1

Introduction 1

LIB99 is a library of subroutines and functions that can be called from both FORTRAN Version 1 and FORTRAN Version 2 on any CYBER 180 model. On the Model 990 of the CYBER 180 the subroutines and functions provide access to the vector capabilities of this machine. On other models, the subroutines and functions are executed in scalar mode. The LIB99 library of routines are available automatically; no special mention of the library is needed on the compilation or execution commands. The LIB99 routines can (among other things):

Perform basic vector arithmetic

Perform basic matrix algebra

Solve linear systems of equations using both direct and iterative methods

Compute Fast Fourier Transforms (FFTs)

Sort lists

Compute eigenvalues and eigenvectors

LIB99 also contains some routines from the Basic Linear Algebra Subroutines (BLAS) package, coded especially for the CYBER 180 Series machines.

LIB99 Terminology

The LIB99 subroutine and function descriptions include terms that may be unfamiliar to a FORTRAN Version 1 user (for example, scalar, vector, stride, and matrix). These terms, as well as array subscript conventions used in this manual, are described below.

Scalars

A mathematician uses the term scalar to describe a set of numbers or variables in which the number of elements is exactly one. A FORTRAN programmer uses the term scalar to describe any single value. For instance, a scalar can be a constant, such as 2.5, or it can be a non-dimensioned variable, such as the real variable ALFA, which is often referred to as a scalar variable. A scalar can also be an array element, such as Y(2,3) or V(J).

Vectors and Strides

To a mathematician, a vector of length N is a set of N numbers uniquely defining a distance and a direction in an N-dimensional space. A FORTRAN programmer generally uses the term vector of length N to mean any one-dimensional set of N numbers. In this manual, both types of references to vectors occur. The former is used in the formulation of algorithms, usually under the heading Method, and as a way of identifying subsets of arrays. In the latter case, a vector of length N means N equidistant locations in memory. The vector is completely defined by its length N in conjunction with a starting address and a stride. The starting address is represented by a specific array element, while the stride is a signed integer specifying the distance and direction in memory between consecutive vector elements. Some examples of vectors that are subsets of a one-dimensional array A:

Starting address:	A(0)
Stride:	1
Vector length:	N
Vector elements:	A(0), A(1), A(2), . . . , A(N-2), A(N-1)

```
Starting address:   A(4)
Stride:             2
Vector length:      3
Vector elements     A(4), A(6), A(8)

Starting address:   A(1)
Stride:             -1
Vector length:      K+2
Vector elements:    A(1), A(0), A(-1), . . . , A(1-K), A(-K)
```

A vector may be identified as a subset of a multi-dimensional array. For instance, the elements stored in a row of a two-dimensional array may be described as a vector. The value of the stride would then be the size of the first dimension, or column length, of that array. This is because FORTRAN uses columnwise storage for two-dimensional arrays, and therefore the distance in memory between consecutive elements in a row is exactly the number of words required to store one column. An example of a vector that is a subset of the two-dimensional array B:

```
DIMENSION statement:    DIMENSION B(0:99, 0:49)
Starting address:       B(J,0)
Stride:                 100
Vector length:          50
Vector elements:        B(J,0), B(J,1), B(J,2), ..., B(J,48), B(J,49)
```

Matrices

A matrix is a two-dimensional mathematical entity consisting of elements arranged in columns and rows. Its shape and size are defined by its dimensions; that is, by two integers specifying the number of elements in each column and row, respectively. In this manual, square brackets are used to enclose corresponding dimension values when defining matrices:

```
[number of rows, number of columns]
```

or equivalently,

```
[column length, row length]
```

As an example, the following arrangement of six elements can be referred to as a matrix of dimension [2,3] or simply as a [2,3] matrix:

$$a_{00}\ a_{01}\ a_{02}$$
$$a_{10}\ a_{11}\ a_{12}$$

Note that the specification of where in the matrix a given element belongs is incorporated as a two-part subscript in the element name; a_{01} belongs in row 0 and column 1, a uniquely defined location. The enumeration of rows and columns may vary, but in this manual we will almost always choose to start with zero, defining the subscript for the element in the upper left-hand corner as 00. Note that the subscript comprises a pair of numbers, the first being the row number and the second being the column number.

A matrix whose elements are numbers (rather than rabbits or something else) can be mapped onto a two-dimensional FORTRAN array, provided that the array is large enough to contain the matrix. It is not necessary that the array has dimension sizes matching the matrix dimensions, nor is it necessary that the DIMENSION statement (or type statement) defining the array use the same row and column enumeration as is done for the matrix. The [2,3] matrix described above can, for instance, be mapped in any one of the three ways below:

DIMENSION X(0:1, 0:2)	DIMENSION Y(3,4)	DIMENSION Z(0:4, 0:4)
a_{00} a_{01} a_{02}	a_{00} a_{01} a_{02} ***	*** *** *** *** ***
a_{10} a_{11} a_{12}	a_{10} a_{11} a_{12} ***	*** a_{00} a_{01} a_{02} ***
	*** *** *** ***	*** a_{10} a_{11} a_{12} ***
		*** *** *** *** ***
		*** *** *** *** ***
The element a_{00} is mapped onto X(0,0) of array X.	The element a_{00} is mapped onto Y(1,1) of array Y.	The element a_{00} is mapped onto Z(1,1) of array Z.

Assume that you have computed the elements of the [2,3] matrix above, and that you want to copy the result from array X, where it is currently stored, into the location in array Y indicated above. The LIB99 subroutine needed for this purpose is MXMOVF, and the two dimension numbers to use as the first two parameters are 2 and 3, respectively. The source array is X, and the associated size of the first dimension, or column length, is 2. The target array is Y and the associated size of the first dimension is 3. Therefore, the call:

```
CALL MXMOVF(2, 3, X, 2, Y, 3)
```

implies that the upper left-hand corner of the matrix is stored as the first element of array X in location X(0,0). Similarly, the execution of the CALL statement will copy the matrix in such a way as to map the element a_{00} onto the first element of array Y; that is, onto location Y(1,1). If you want to copy the matrix into the indicated location of array Z, you must include the subscripts indentifying the target location for the upper left-hand corner of the matrix. This is because that location does not coincide with the location of the first element of array Z, which is Z(0,0), but rather with that of the seventh, which is Z(1,1). Therefore, in this case, the proper call is:

```
CALL MXMOVF(2, 3, X, 2, Z(1,1), 5)
```

Array Subscript Conventions

A general description of arrays can be found in the FORTRAN Language Definition manual (publication number 60485913). This includes a description of how arrays are stored in the computer memory, as well as how subscripts are used to identify array locations.

In this documentation we have adopted the convention that all arrays are defined with the lower bound equal to zero for all subscripts, and that all subscripts between the lower and upper bound correspond to actual array element locations. Thus, a one-dimensional array X of length 100 is assumed to be defined somewhere as X(0:99). Similarly, a two-dimensional array A with column length N will be referenced as if it had appeared in a DIMENSION or type statement as A(0:N-1,0:M), where N and M are either dummy arguments or symbolic constants.

A special array notation has been developed to describe a vector and accompanying stride in a compact way. The array subscript of the last vector element, representing the ending address, must first be computed. A vector that is a subset of the one-dimensional array A is described as:

```
A( first subscript : last subscript : stride )
```

For example, this compact notation is used to describe the three vectors defined above under Vectors and Strides:

Starting Address	Stride	Length	Compact Array Notation
A(0)	1	N	A(0:N-1:1)
A(4)	2	3	A(4:8:2)
A(1)	-1	K+2	A(1:-K:-1)

To use the compact array notation to describe vectors that are subsets of multi-dimensional arrays, we must replace the term stride with subscript increment, a value specifying by how much a particular array subscript needs to be incremented in order to find the next vector element. For a one-dimensional array there is no difference between the two terms. Thus, the more general notation is:

```
A(first subscript : last subscript : subscript increment).
```

An omitted subscript increment defaults to 1, so that A(0:N-1:1) and A(0:N-1) describe the same vector.

This array notation lends itself well to describing regularly shaped subsets of two-dimensional arrays. For example, the areas occupied by the [2,3] matrix in the three cases above can be described as:

Matrix in Array X, Y, and Z	Array Notation
The element a_{00} is mapped onto X(0,0) of array X.	X(0:1,0:2)
The element a_{00} is mapped onto Y(1,1) of array Y.	Y(1:2,1:3)
The element a_{00} is mapped onto Z(1,1) of array Z.	Z(1:2,1:3)

The work done by the subroutine MXMOVF in the two examples above can now be expressed as moving elements from one part of an array to a part of another array:

```
CALL MXMOVF(2, 3, X, 2, Y, 3)
```

The elements in the array section X(0:1,0:2) are moved to Y(1:2,1:3).

```
CALL MXMOVF(2, 3, X, 2, Z(1,1), 5)
```

Similarly, the elements in the array section X(0:1,0:2) are moved to Z(1:2,1:3).

Executing Your Program With the LIB99 Library

Although you compile your program containing references to LIB99 subroutines and function as you would any FORTRAN program, you must obtain the LIB99 library when executing your program. To do this, specify LIBRARY=$SYSTEM.COMMON.LIB99 on the EXECUTE_TASK command. For example, to execute a program containing calls to the LIB99 library whose binary file is BFILE1, use the following command:

```
EXECUTE_TASK  F=BFILE1  L=$SYSTEM.COMMON.LIB99
```

You can also add the library each time you logon by including the following command in your user prolog:

```
SET_PROGRAM_ATTRIBUTE  ADD_LIBRARY=$SYSTEM.COMMON.LIB99
```

Externals

The LIB99 subroutines and functions themselves may reference other routines. The routines they call, if any, are listed for each LIB99 subroutine or function under the heading Externals. The external routines may be other routines in LIB99 or they may be routines from the standard FORTRAN runtime library. Mathematical routines are described in the Math Library for NOS/VE manual.

Dynamic Space

Calls to some of the functions and subroutines in LIB99 require an allocation of memory for temporary "scratch" vectors. This memory area is called dynamic space and the number of words needed, if any, is indicated for each routine under the heading Dynamic Space. A word is equal to eight bytes of memory. The dynamic space is freed upon return from the routine.

When a LIB99 routine calls another routine, as indicated under the heading Externals in the corresponding documentation, the amount of dynamic space reported refers to that which is used by the documented routine itself, and does not include any that may be allocated by the called (external) routines.

NOTE

Since a certain amount of overhead is associated with calling a subroutine or function, it is recommended that vectors of length at least 50 be used in calls to the basic vector arithmetic routines and that matrices for which the size of the first dimension (column length) is at least 10 be used in calls to the matrix algebra routines.

BLAS Conventions

The following LIB99 routines are direct translations from the public domain BLAS (Basic Linear Algebra Subroutines) package:

SASUM
SAXPY
SCOPY
SDOT
SNRM2
SSCAL
SSWAP

The BLAS routines generally perform simple vector operations, where each vector is defined in terms of an array address and accompanying stride. If the value of the stride associated with an array A is zero, then A will be interpreted as a scalar constant or variable, or array element. It will be broadcast in the vector operation as a vector with all elements identical to each other.

Positive strides are interpreted in the same manner as they are in the other LIB99 routines, as described in the Introduction. However, negative strides have a special meaning in the BLAS routines, as illustrated by the following call to SAXPY:

```
CALL SAXPY(N, A, X, LX, Y, LY)
```

In this call, LX and LY are the strides associated with the arrays X and Y, respectively. Assume that LX is positive and LY negative. The N elements of X would then be referenced in order of increasing subscript values:

```
X(J*LX), J=0, 1, ..., N-2, N-1
```

or

```
X(0), X(LX), ..., X((N-2*LX), X((N-1)*LX)
```

In contrast, the N elements of Y would be referenced in reverse order, starting from the end; that is, in order of decreasing subscript values:

```
Y(-J*LY), J=N-1, N-2, ..., 1, 0
```

or

```
Y(-(N-1)*LY), Y(-(N-2)*LY), ..., Y(-LY), Y(0)
```

To further illustrate how strides are interpreted by the BLAS routines, the equivalent scalar FORTRAN code for the SAXPY subroutine is given below:

```
      SUBROUTINE SAXPY(N, A, X, LX, Y, LY)
      REAL X(0:*), Y(0:*)
      IF (N .LE. 0) RETURN
      JX = 0
      IF (LX .LT. 0) JX=-(N-1)*LX
      JY=0
      IF (LY .LT. 0) JY=-(N-1)*LY
      DO 10 J= 1, N
      Y(JY) = A*X(JX) + Y(JY)
      JX = JX + LX
      JY = JY + LY
10    CONTINUE
      RETURN
      END
```

LIB99 Subroutines and Functions 2

LIB99 Subroutines and Functions 2

The following table lists the LIB99 subroutines and functions, modules or entry points, and brief descriptions of the subroutine or function. The detailed descriptions of the subroutines and functions are listed in alphabetical order by their title name following this table.

Title	Modules or Entry Points	Description
CONV	CONV, CORR, FILTGS, SOP	Convolution and correlation
EVBAK	EVBAK	Backtransform eigenvectors
EVIQL	EVIQL	Solve standard symmetric tridiagonal eigenvalue problem
EVRED	EVRED	Reduce general eigenvalue problem to standard
EVRS	EVRS	Solve standard symmetric eigenvalue problem
EVRSG	EVRSG	Solve general symmetric eigenvalue problem
FFT1D	FFT1D	Base 2/4 Fourier Transform
F64TR4	F64TR4	CYBER real to IBM REAL*4
F64TSR	F64TSR, F64TSRN	CYBER real to IEEE short real
F64TVF	F64TVF	CYBER real to VAX 32-bit real
GENSPD	GENSPD	Generate symmetric positive definitive matrix
ITPACKV	JCG, JSI, SOR, SSORCG, SSORSI, RSCG, RSSI	Iterative linear equation solvers for sparse matrices
I64TSI	I64TI2, I64TSI	CYBER integer to IEEE short integer
I64TWI	I64TI4, I64TWI	CYBER integer to IEEE word integer
LVCOMP	LIVEQ, LIVGE, LIVLT, LIVNE	Integer vector compare
LVCOMP	LRVEQ, LRVGE, LRVLT, LRVNE	Real vector compare
MOVB	MOVB	Move byte string
MXADDF	MXADDF	Full real matrix add
MXCMP	MXCMP	Full real matrix compare
MXENRM	MXENRM	Compute the Euclidean norm of a matrix
MXEQ	MXEQ	Full matrix bit-by-bit compare
MXGEFS	MXGEFS	Direct linear equation solver (Gaussian elimination)
MXINVU	MXINVU	Upper triangular matrix inverse
MXMAB	MXMAB	Full real matrix multiply

(Continued)

Title	Modules or Entry Points	Description
MXMOVF	MXMOVF	Full matrix move
MXMOVU	MXMOVU	Upper triangular matrix move
MXMUTU	MXMUTU	U(transp)*U, U upper triangular matrix
MXMUUT	MXMUUT	U*U(transp), U upper triangular matrix
MXSCAF	MXSCAF	Full real matrix scaling
MXSUBF	MXSUBF	Full real matrix add
MXSYUL	MXSYUL	Symmetric matrix UL-decomposition
MXTRAF	MXTRAF	Full matrix transpose
MXTRAU	MXTRAU	Upper triangular matrix transpose
MX1NRM	MX1NRM	Compute the 1-norm of a matrix
MX8NRM	MX8NRM	Compute the infinity-norm of a matrix
QSORT	IQSORT, QSORT	Vector quicksort (scalar algorithm)
R4TF64	R4TF64	IBM REAL*4 to CYBER real
SASUM	SASUM	Real vector absolute sum (BLAS)
SAXPY	SAXPY	Real vector a*X+Y (BLAS)
SCOPY	SCOPY	Real or integer vector move (BLAS)
SDOT	SDOT	Real vector dot product (BLAS)
SITI64	I2TI64, SITI64	IEEE short integer to CYBER integer
SNRM2	SNRM2	Real Euclidean vector length (BLAS)
SRTF64	SRNTF64, SRTF64	IEEE short real to CYBER real
SSCAL	SSCAL	Real vector scaling (BLAS)
SSWAP	SSWAP	Real or integer vector swap (BLAS)
TRED2	TRED2	Tridiagonalize symmetric matrix
VABS	IVABS, RVABS	Vector absolute value
VADD	IVADD, ISADDV, IVADDS, IVADDV	Integer vector add
VADD	RVADD, RSADDV, RVADDS, RVADDV	Real vector add
VAND	VAND, SANDV, VANDS, VANDV	Logical vector AND
VAXPY	RVAXMY, RVAXPY	Linked triad: a*X-Y, a*X+Y
VDIV	IVDIV, ISDIVV, IVDIVS, IVDIVV	Integer vector divide
VDIV	RVDIV, RSDIVV, RVDIVS, RVDIVV	Real vector divide

(Continued)

Title	Modules or Entry Points	Description
VFILL	VFILL	Vector fill
VFLOAT	VFLOAT	Vector convert integer to real
VFTF64	VFTF64	VAX 32-bit real to CYBER real
VGATHER	VGATHP, VGATHR, VGATHR1	Vector periodic and random gather
VIFIX	VIFIX	Vector convert real to integer
VIOR	VIOR, SIORV, VIORS, VIORV	Logical vector inclusive OR
VMASK	VMASK0, VMASK1, VMASK2	Merge two vectors
VMASUM	VMASUM	Moving window vector absolute sum
VMUL	IVMUL, ISMULV, IVMULS, IVMULV	Integer vector multiply
VMUL	RVMUL, RSMULV, RVMULS, RVMULV	Real vector multiply
VRANF	VRANF	Vector random number generator
VSCATTER	VSCATP, VSCATR, VSCATR1	Vector periodic and random scatter
VSHFC	VSHFC, SSHFCV, VSHFCS, VSHFCV	Circular vector shift
VSUB	IVSUB, ISSUBV, IVSUBS, IVSUBV	Integer vector subtract
VSUB	RVSUB, RSSUBV, RVSUBS, RVSUBV	Real vector subtract
VSUM	RVASUM, RVSUM	Real vector (absolute) sum
VXOR	VXOR, SXORV, VXORS, VXORV	Logical vector exlusive OR
WITI64	I4TI64, WITI64	IEEE word integer to CYBER integer

Subroutine CONV

Purpose

To compute the convolution of two vectors. Depending on the relative length of the vectors, the process is often referred to as either (auto)correlation or sum-of-products.

Format

```
CALL CONV   ( A , NA , F , NF , IOFF , IREV ,    R , NR )
CALL CORR   ( A , NA , F , NF , IOFF , IREV ,    R , NR )
CALL FILTGS ( F , NF , A , NA , R    ,   NR , IREV , IOFF )
CALL SOP    ( A , NA , F , NF , IOFF , IREV ,    R , NR )
```

Parameters

A

Array serving as one of the input operands in the convolution. Must be of type real. When A and F are of substantially different lengths, A is typically the longer of the two. The content of A is preserved by this routine.

NA

Length of array A. Must be of type integer. No element before A(0) or after A(NA-1) is used in the computations. NA must be at least as large as NF-ABS(IOFF).

F

Array serving as one of the input operands in the convolutions. Must be of type real. This should be the shorter one if A and F have substantially different lengths. The content of F is preserved by this routine.

NF

Length of array F. Must be of type integer. NF must not exceeed NA+ABS(IOFF). Elements outside the range F(0) through F(NF-1) are not used.

IOFF

Offset value. Must be of type integer. Only the absolute value of IOFF is used, and it specifies how many steps to the left of the beginning of A the F vector should be aligned when the first result, R(0), is computed. The absolute value of IOFF must be smaller than NF. See Method for more information on this parameter.

IREV

Flag. Must be of type integer. When IREV=0 the F vector is used in a normal manner. When IREV≠0 the F vector is reversed prior to usage. The reversal is either conceptual or uses dynamic space, so that the content of F is not affected by this procedure.

R

Result vector. Must be of type real.

NR

Length of array R. Must be of type integer. If NR exceeds NA+ABS(IOFF), the last NR-NA-ABS(IOFF) locations of R are zeroed.

Externals

None

Dynamic Space

Normally 573 words. If IREV≠0 and the inner product method is chosen (or forced through calling CORR), then an extra NF words are used.

Method

If IOFF=0, NF≤NA, and NR≤(NA-NF+1), then the computations are performed either by means of NR dot product operations of length NF, or NF linked triads of length NR. The choice of method is mainly dictated by the relative lengths of R and F. When CONV or FILTGS is called, the routine estimates internally which method is faster and selects that one. Thereby, a more precise method involving the exact values of IOFF, NF, NA and NR is used. By calling SOP the linked triad method is forced, and by calling CORR the dot product method is forced. Note that although there is no mathematical difference between the two methods, there is a numerical difference since the arithmetic operations are performed in different order for the two cases.

The process can be pictorially described by first imagining the A vector stretched out horizontally, with the F vector lying right on top of it. Aligned directly over A(0) should be F(JF), where JF=ABS(IOFF). The dot product to be computed for the first result element, R(0), is then represented by the F and A elements that overlap in this picture. More precisely, R(0)=(SUM(A(J)*F(JF+J)),J=0,...,NF-ABS(IOFF)-1).

Subsequent elements of R are computed by moving the F vector to the right one step at a time, and computing the corresponding dot products. When F does not extend beyond the limits of A, the dot product consists of exactly NF elements; otherwise, it is shorter.

Example

```
PROGRAM MAIN
REAL ARRAY1(0:99), ARRAY2(0:49), RESULT(0:74)
LENGTH1=90
LENGTH2=50
IOFF=0
IFLAG=0
LENGTHR=75
.
.
.
CALL CONV(ARRAY1, LENGTH1, ARRAY2, LENGTH2, IOFF, IFLAG, RESULT, LENGTHR)
END
```

In this example the CONV subroutine is called to compute the convolution of the first 90 elements of ARRAY1 and all of the elements of ARRAY2. The result is returned in array RESULT.

Subroutine EVBAK

Purpose

To determine the eigenvectors of a real symmetric generalized eigensystem A*X = lambda*B*X, where A and B are real symmetric matrices and B is positive definite, by back transforming the eigenvectors of the real symmetric standard eigenproblem G*Z = lambda * Z formed in EVRED.

Format

CALL EVBAK (N , U , KU , D , NEV , Z , KZ)

Parameters

N

The order of the matrices U and Z. Must be of type integer.

U

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the upper triangular matrix U appearing in the equation which this routine must solve, namely Z = U(transpose)*X. Must be of type real. U should be the upper triangular factor of the Cholesky decomposition of the matrix B appearing in the formulation of the real symmetric generalized eigenvalue problem A*X = lambda*B*X. The matrix U is normally produced as the result of a call to EVRED, which in turn calls MXSYUL to perform the actual decompositition. The contents of U are not altered by this routine.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}\ u_{01}\ u_{11}\ u_{02}\ u_{12}\ u_{22}\ u_{03}\ \cdots$$

A value of N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the subdiagonal elements are ignored.

D

Array (element) indicating the address of the input array containing the reciprocal values of the N diagonal elements of the upper triangular matrix U. Must be of type real. Normally the contents of D, as well as those of U, are produced as a result of a call to EVRED. The contents of D are not altered by this routine.

NEV

The number of eigenvectors to be back transformed.

Z

Array (element) indicating the address of the upper left-handcorner of a two-dimensional input/output matrix. Must be of type real. EVBAK is designed to be the last step in solving the real symmetric generalized eigenvalue problem. Thus, on input Z must contain the N eigenvectors of the real symmetric matrix G, where G was formed in EVRED and Z in EVIQL, and its contents must therefore adhere to the format explicitly specified in the documentation of those two LIB99 routines. A more precise definition is given under Method.

On output, the first NEV columns of Z will contain the NEV back transformed eigenvectors; that is, the actual eigenvectors of the generalized eigenvalue problem.

Parameters (Continued)

KZ

Size of the first dimension (column length) of array Z. Must be of type integer. Must be greater than or equal to N.

Externals

SDOT

Dynamic Space

None

Method

The real symmetric generalized eigenvalue problem is defined by the equation:

$$A * X = \text{lambda} * B * X \qquad (1)$$

where A and B are real symmetric matrices and B is positive definite. X represents the square matrix whose columns are the eigenvectors of the thus defined eigensystem, and lambda is the diagonal matrix whose elements are the corresponding eigenvalues.

By performing a Cholesky decomposition of the matrix B we obtain B = U*L, where U is an upper triangular matrix and L is its transpose. By pre-multiplying equation (1) above with the inverse of U, and inserting the identity matrix in the form of the matrix product L(inverse)*L, or L(inverse)*U(transpose), we obtain the equation

$$U^{-1} * A * L^{-1} * U^{T} * X = \text{lambda} * U^{T} * X \qquad (2)$$

Since it can be shown that the matrix G = U(inverse)*A*L(inverse) is similar to B(inverse)*A, and since the theory of eigensystems tells us that similar matrices have identical eigenvalues, the original real symmetric generalized eigenvalue problem can now be restated as follows:

$$B = U * L = U * U^{T} \qquad (3)$$

$$G = U^{-1} * A * L^{-1} \qquad (4)$$

$$G * Z = \text{lambda} * Z \qquad (5)$$

$$Z = U^{T} * X \qquad (6)$$

The routine EVBAK solves the last of these equations for X, thereby transforming, in place, the eigenvectors of the real symmetric standard eigenvalue problem (5), contained in Z, into those of the real symmetric generalized eigenvalue problem (1). The bulk of the work is performed by means of calls to the LIB99 routine SDOT.

Subroutine EVIQL

Purpose

To find all eigenvalues and, optionally, all eigenvectors of a real symmetric tridiagonal matrix. The complete eigensystem of a general real symmetric matrix can also be determined, provided the matrix has first been tridiagonalized by calling TRED2.

Format

CALL EVIQL (N , Z , KZ , D , S , T , W , IVEC , ITMAX , NEV)

Parameters

N

The order of the eigensystem. Must be of type integer.

Z

Array (element) indicating the address of the upper left-hand corner of a two-dimensional input/output array. Must be of type real. This parameter is ignored when IVEC=0 (see below). When IVEC=1 there are two cases:

1. If Z contains the identity matrix, then the eigenvectors of the tridiagonal matrix contained in the arrays D and S are determined. The identity matrix is defined by the requirement that Z(J,K)=1.0 when J=K, but zero otherwise.

2. If Z contains the transformation matrix produced in the reduction of a real symmetric matrix to the tridiagonal matrix contained in the arrays D and S, by means of a call to TRED2, then the eigenvectors of the original real symmetric matrix are determined.

In both of these cases, on return from EVIQL, Z will contain in its first NEV columns the NEV eigenvectors associated with the NEV successfully computed eigenvalues.

KZ

Size of the first dimension (column length) of array Z. Must be of type integer. This parameter is ignored when IVEC=0. When IVEC=1, KZ must be greater than or equal to N.

D

Input/output array of length at least N words. Must be of type real. On input D should contain the diagonal elements of the symmetric tridiagonal input matrix. On output D will in its first NEV positions contain the NEV successfully determined eigenvalues, ordered ascendingly with respect to algebraic value.

S

Input array of length at least N words. Must be of type real. S should contain in its first N-1 positions the subdiagonal elements of the tridiagonal input matrix. On return from EVIQL the contents of S will have been destroyed.

T

Scratch array of length at least 2*N words. May be of any type except character. This parameter is ignored when IVEC=0.

W

Scratch array of length at least N words. Can be of any type except character.

Parameters (Continued)

IVEC

Input selection parameter. Must be of type integer. A value of 0 requests eigenvalues only. A value of 1 requests eigenvalues and eigenvectors.

ITMAX

User-supplied value limiting the number of iterations allowed in the determination of any given eigenvalue. Must be of type integer. If the value 0 is supplied, a default value is used (currently 30). If for the Jth eigenvalue ITMAX iterations is not enough to achieve convergence, the algorithm is discontinued and the output parameter NEV is assigned the value J-1.

NEV

Output parameter of type integer. See discussion of ITMAX above.

Externals

QSORT, RSMULV, RVAXMY, RVAXPY, SQRT, SSWAP

Dynamic Space

None

Method

The routine implements the implicit QL-algorithm, as described in Wilkinson-Reinsch: Linear Algebra, Handbook for Automatic Computation, Vol.2, Springer-Verlag, Berlin, 241-248 (1971). That algorithm is highly recursive, and does not lend itself easily to vectorization. The process of "forming vectors" (see reference) is vectorized, but the rest of the computations in the main part of the algorithm are done in scalar mode. When all eigenvalues and eigenvectors have been determined, they are ordered ascendingly with respect to the algebraic value of the eigenvalues. This is taken care of by a call to QSORT, followed by a rearrangement of the eigenvectors according to the index list which is produced by QSORT.

Subroutine EVRED

Purpose

To reduce the real symmetric generalized eigenproblem A*X = lambda*B*X, where A and B are real symmetric matrices and B is positive definite, to the real symmetric standard eigenproblem G*Z = lambda * Z.

Format

CALL EVRED (N , A , KA , B , KB , G , KG , U , KU , D , W , IERR)

Parameters

N

The order of the eigensystem. Must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric matrix A. Must be of type real. The contents of A will not be altered by this routine unless A and G are the same array, which is allowed (see restrictions for KA and KG).

KA

Storage mode indicator for A. Must be of type integer. If A and G are the same array, then either KA=KG=1 or KA.GE.KG.GE.N must hold. If A and G are distinct, then the case KG.GT.KA.GE.N is also permitted.

A value of 1 indicates that the columns of the upper triangular part of A are stored linearly, thus omitting the subdiagonal elements:

$$a_{00}\text{-}a_{01}\text{-}a_{11}\text{-}a_{02}\text{-}a_{12}\text{-}a_{22}\text{-}a_{03}\text{-}\cdots$$

A value of N indicates full storage mode; that is, KA is the size of the first dimension (column length) of the two-dimensional array A. In this case the subdiagonal elements are ignored and will not be altered by this routine.

B

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric positive definite matrix B. Must be of type real. The contents of B are not altered by EVRED unless B and U are the same array, which is permitted when KB=KU.

KB

Storage mode indicator for B. Must be of type integer. If B and U are the same array, then KB=KU must hold. A value of 1 indicates that the columns of the upper triangular part of B are stored linearly, thus omitting the subdiagonal elements:

$$b_{00}\text{-}b_{01}\text{-}b_{11}\text{-}b_{02}\text{-}b_{12}\text{-}b_{22}\text{-}b_{03}\text{-}\cdots$$

A value of N indicates full storage mode; that is, KB is the size of the first dimension (column length) of the two-dimensional array B. In this case the subdiagonal elements are ignored and will not be altered by this routine.

Parameters (Continued)

G

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric output matrix G created by EVRED. Must be of type real. G is formed as the triple matrix product U(inverse)*A*L(inverse), where U is the upper triangular matrix defined below, and L is its transpose. G may be the same array as A provided that KG and KA satisfy certain conditions (see respective descriptions).

KG

Storage mode indicator for G. Must be of type integer. If A and G are the same array, then either KA=KG=1 or KA.GE.KG.GE.N must hold. If A and G are distinct, then the case KG.GT.KA.GE.N is also permitted.

A value of 1 indicates that the columns of the upper triangular part of G are stored linearly, thus omitting the subdiagonal elements:

$$g_{00}\text{-}g_{01}\text{-}g_{11}\text{-}g_{02}\text{-}g_{12}\text{-}g_{22}\text{-}g_{03}\text{-}\cdots$$

A value of N indicates full storage mode; that is, KG is the size of the first dimension (column length) of the two-dimensional array G. In this case the subdiagonal elements are ignored and will not be altered by this routine.

U

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the upper triangular matrix U resulting from the Cholesky decomposition of B, as performed by MXSYUL when called by EVRED. Must be of type real. The matrix U is defined by the matrix equation B = U*L, where U is an upper triangular matrix and L is its transpose. U may be the same array as B, provided that KU=KB.

KU

Storage mode indicator for U. Must be of type integer. If U and B are the same array, then KU=KB must hold.

A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}\text{-}u_{01}\text{-}u_{11}\text{-}u_{02}\text{-}u_{12}\text{-}u_{22}\text{-}u_{03}\text{-}\cdots$$

A value of N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the subdiagonal elements are ignored and will not be altered by this routine.

D

Array (element) indicating the address of an output array of length at least N words. Must be of type real. On return from EVRED, D will contain the reciprocal values of the N diagonal elements of the upper triangular matrix U.

W

Workspace of length N words. May be of any type except character.

IERR

Error flag set by this routine. Must be of type integer. If B is nonpositive definite, IERR will contain a nonzero value on return from EVRED. Normal return is IERR = 0.

Externals MXSYUL, RSMULV, RVAXPY, SCOPY, VGATHP

Dynamic Space None

Method The call to EVRED requires two input matrices, A and B, and produces two output matrices, G and U, as well as an auxiliary array, D. The work done by EVRED is intended to provide the first step in the solving of the real symmetric generalized eigenvalue problem, which schematically can be depicted in the following manner:

a) Problem - Solve for both lambda and X: $A*X = lambda*B*X$

b) Perform UL decomposition of B: $A*X = lambda*U*L*X$

(B=UL, where U is an upper triangular matrix and L is its transpose. U is computed by means of a call to MXSYUL.)

c) Pre-multiply with the inverse of U: $U^{-1}*A*X = lambda*L*X$

d) Embed the identity matrix $L^{-1}L$: $U^{-1}*A*L^{-1}*L*X = lambda*L*X$

e) Substitute $G = U^{-1} * A*L^{-1}$ and $U^{T} = L$: $G * (U^{T}*X) = lambda * (U^{T}*X)$

Since it can be shown that the matrix G = U(inverse)*A*L(inverse) is similar to B(inverse)*A, and since the theory of eigensystems tells us that similar matrices have identical eigenvalues, the original real symmetric generalized eigenvalue problem can now be restated as follows:

1. Compute U defined by: $B = U * L = U * U^{T}$

2. Compute G defined by: $G = U^{-1} * A * L^{-1}$

3. Solve the real symmetric standard eigenvalue problem: $G * Z = lambda * Z$

4. Backtransform the eigenvectors found by solving for X in: $Z = U^{T} * X$

The work done by EVRED consists of computing the upper triangular part of the symmetric matrix G, the upper triangular matrix U, and the reciprocal values of the diagonal elements of U, as returned in array D. This corresponds to the steps numbered 1 and 2 above. Step 3 can be performed by calling the LIB99 routines TRED2 and EVIQL, while step 4 can be handled by the subroutine EVBAK, also in LIB99.

Subroutine EVRS

Purpose

This subroutine calls two LIB99 subroutines to determine the eigenvalues and eigenvectors of the real symmetric standard eigenproblem A*X = lambda*X, where A is a real symmetric matrix. Here X represents the solution matrix whose columns consists of the eigenvectors of A, while lambda represents the diagonal matrix whose elements are the eigenvalues of A.

Format

CALL EVRS (N , A , KA , E , IVEC , Z , KZ , W , NEV)

Parameters

N

The order of the eigenproblem. Must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric matrix A. Must be of type real. A and Z may be the same array provided that KA=KZ. The contents of A are not altered by EVRS unless Z is the same array as A.

KA

Storage mode indicator for A. Must be of type integer. If A and Z are the same array, then KA=KZ must hold. This is possible only when KA.GE.N. A value of 1 indicates that the columns of the upper triangular part of A are stored linearly, thus omitting the subdiagonal elements:

$$a_{00}-a_{01}-a_{11}-a_{02}-a_{12}-a_{22}-a_{03}-\cdots$$

A value of N indicates full storage mode; that is, KA is the size of the first dimension (column length) of the two-dimensional array A. In this case the subdiagonal elements are ignored and will not be altered by this routine, unless A and Z are the same array which is permitted if KA=KZ.

E

Output array of length at least N words. Must be of type real. On return from EVRS, the first NEV locations of E will contain the NEV successfully determined eigenvalues, ordered ascendingly with respect to algebraic value.

IVEC

Input selection parameter. Must be of type integer. A value of 0 requests eigenvalues only. A value of 1 requests eigenvalues and eigenvectors.

Z

Array (element) indicating the address of the upper left-hand corner of a two-dimensional output array. Must be of type real. Array Z is effectively a scratch array when IVEC=0.

If IVEC = 1, the first NEV columns of Z will on return from EVRS contain the NEV eigenvectors associated with the NEV successfully determined eigenvalues. Z and A may be the same array when KZ = KA.

Parameters (Continued)

KZ

Size of the first dimension (column length) of array Z. Must be greater than or equal to N. Must be of type integer.

W

Scratch array of length at least 4N words. May be of any type except character.

NEV

Return parameter of type integer. On return from EVRS the value of NEV will be set to the number of successfully computed eigenvalues. Normal return is NEV = N.

Externals

EVIQL, TRED2

Dynamic Space

None

Method

EVRS proceeds by making calls to the LIB99 routines TRED2 and EVIQL. Additional details can be found in the documentation of these routines.

Subroutine EVRSG

Purpose

To solve the real symmetric generalized eigenvalue problem A*X = lambda*B*X, where A and B are real symmetric matrices and B is positive definite. Eigenvalues and, optionally, eigenvectors are computed.

Format

CALL EVRSG (N , A , KA , B , KB , E , IVEC , Z , KZ , W , NEV)

Parameters

N

The order of the eigensystem. Must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric matrix A. Must be of type real. This routine will destroy the contents of array A. Array A may be the same as Z, provided that KA=KZ.

KA

Storage mode indicator for A. Must be of type integer. If A and Z are the same array, then KA=KZ must hold. This is possible only when KA.GE.N.

A value of 1 indicates that the columns of the upper triangular part of A are stored linearly, thus omitting the subdiagonal elements:

$$a_{00}-a_{01}-a_{11}-a_{02}-a_{12}-a_{22}-a_{03}-\cdots$$

A value of N indicates full storage mode; that is, KA is the size of the first dimension (column length) of the two-dimensional array A. In this case the subdiagonal elements are ignored and will not be altered by this routine, unless A and Z are the same array, which is permitted if KA=KZ.

B

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric positive definite input matrix B. Must be of type real. On return the upper triangular part of B will have been altered to contain the upper triangular part of the upper triangular matrix U, produced by MXSYUL when called by EVRED to perform a Cholesky decomposition of B.

KB

Storage mode indicator for B. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of B are stored linearly, thus omitting the subdiagonal elements:

$$b_{00}-b_{01}-b_{11}-b_{02}-b_{12}-b_{22}-b_{03}-\cdots$$

A value of N indicates full storage mode; that is, KB is the size of the first dimension (column length) of the two-dimensional array B. In this case the subdiagonal elements are ignored and will not be altered by this routine.

E

Output array of length at least N words. Must be of type real. On return from EVRSG, E will in its first N locations contain the NEV successfully determined eigenvalues, ordered ascendingly with respect to algebraic value.

Parameters (Continued)

IVEC

Input selection parameter. Must be of type integer. A value of 0 requests eigenvalues only. A value of 1 requests eigenvalues and eigenvectors.

Z

Array (element) indicating the address of the upper left-hand corner of a two-dimensional output array. Must be of type real. When IVEC=0 array Z is effectively a scratch array.

If IVEC = 1, the first NEV columns of Z will on return from EVRSG contain the NEV eigenvectors associated with the NEV successfully determined eigenvalues. Z and A may be the same same array, provided that KZ = KA (implying KA.GE.N).

KZ

Size of the first dimension (column length) of array Z. Must be greater than or equal to N. Must be of type integer.

W

Scratch array of length at least 5*N words. May be of any type except character.

NEV

Return parameter of type integer. If the matrix B, possibly due to rounding errors, proves not to be positive definite, then NEV will be set to -1 and computations discontinued. Otherwise NEV will on return from EVRSG contain the number of successully computed eigenvalues. Normal return is NEV = N.

Externals

EVBAK, EVIQL, EVRED, TRED2

Dynamic Space

None

Method

The method used by EVRSG is to call the four LIB99 routines EVRED, TRED2, EVIQL, and EVBAK. The different steps are described below.

a) Problem - Solve for both lambda and X: $A*X = lambda*B*X$

b) Perform UL decomposition of B: $A*X = lambda*U*L*X$

(B=UL, where U is an upper triangular matrix and L is its transpose. U is computed by means of a call to MXSYUL.)

c) Pre-multiply with the inverse of U: $U^{-1}*A*X = lambda*L*X$

d) Embed the identity matrix $L^{-1}L$: $U^{-1}*A*L^{-1}*L*X = lambda*L*X$

e) Substitute $G = U^{-1} * A*L^{-1}$ and $U^{T} = L$: $G * (U^{T}*X) = lambda * (U^{T}*X)$

Method (Continued)

Since it can be shown that the matrix G = U(inverse)*A*L(inverse) is similar to B(inverse)*A, and since the theory of eigensystems tells us that similar matrices have identical eigenvalues, the original real symmetric generalized eigenvalue problem can now be restated as follows:

1. Compute U defined by: $B = U * L = U * U^{T}$

2. Compute G defined by: $G = U^{-1} * A * L^{-1}$

3. Solve the standard real symmetric eigenvalue problem: $G * Z = lambda * Z$

4. Backtransform the eigenvectors found by solving for X in: $Z = U^{T} * X$

First EVRED is called to compute the real symmetric matrix G and the upper triangular matrix U, corresponding to the steps numbered 1 and 2 above. Step 3 is performed by means of calling TRED2 and EVIQL, while step 4 is handled by a call to EVBAK. Additional details can be found in the documentation of the four mentioned routines.

Subroutine FFT1D

Purpose

To compute the one-dimensional Fast Fourier Transform (FFT) of one or several real or complex vectors whose lengths can be expressed as N=(2**p)*(3**q)*(5**r), where p, q, and r are nonnegative integers. (Currently, q=r=0 is required.)

Format

CALL FFT1D (X , L , N , DTYP , S , TA , TB , XNORM , IOPT , MODE)

Parameters

X

Input/output array of length at least 2*N*L words. Must be of type real or complex (see Remarks). On input, array X should contain the real or complex vector(s) stored as determined by the values of DTYP, XNORM, and MODE. On output, array X contains the transformed vectors. See Remarks for details.

L

The number of one-dimensional transforms to be computed. Must be of type integer. This corresponds to the column length of array X.

N

Number of elements in each vector to be transformed. Must be of type integer. N must be a composite integer of the form N=(2**p)*(3**q)*(5**r), where p, q, and r all are nonnegative integers. N must always be even; that is, p must be nonzero whenever DTYP='RD' (see below).

NOTE

Currently q=r=0 is required; the base-3 and base-5 algorithms are not completed.

DTYP

Character string of length 2 that can take either the value 'CD' or 'RD', meaning complex data or real data, respectively. When DTYP='CD' the input and output vectors are all complex. When DTYP= 'RD' and XNORM is positive, a forward transform of real vectors into conjugate symmetric complex vectors is requested, while DTYP='RD' and XNORM negative requests the inverse transform of conjugate symmetric complex vectors into real vectors. See Remarks for details.

S

Scratch array of the same size as array X. Can be of any type except character.

TA

Array of length at least 2N words when N contains at least one factor of 3 or 5 (that is, when (q+r)>0; see N above) and of length at least N words when N=2**p holds true. Must be of type real. The array is used to hold trigonometric tables necessary to perform the transforms and will be filled by this routine when desired, as specified by the value of IOPT. TA is then never modified. Thus, once created, TA can be used repeatedly in calls with the same value of N.

Parameters (Continued)

TB

Array of length at least N*L/2 words, used to hold an expanded trigonometric table. Must be of type real. Array TB is used only when MODE is odd (MODE=+1 or -1) and N is a multiple of 4. Furthermore, if the routine determines that it is not favorable to use the algorithm requiring TB, then TB is ignored. This evaluation is based on the values of N and L, and TB will not be used if L is too large. (The cutoff is difficult to specify exactly since it depends on N as well. When TB is actually used, the computations proceed at a significantly higher speed, and it is therefore important to allow for its use, especially when L is small.) See Method for more information.

This table can also be used repeatedly.

XNORM

This parameter must be of type real and has two functions:

1. Its sign determines the type of transform. When XNORM is positive, a forward transform is requested. A negative value requests an inverse transform.

2. Its absolute value deals with the normalization of the data. If XNORM equals +1.0 or -1.0, no normalization takes place. Otherwise, the data will be normalized by multiplication with ABS(XNORM).

Two consecutive calls using XNORM=XN1 and XNORM=XN2 recover the original data if XN1*XN2 = -1.0 /FLOAT(N).

IOPT

This parameter must be of type integer and has two functions:

1. Its sign determines whether this routine should compute any Fourier transform at all:

   IOPT>0 creates requested tables and performs the FFT(s).

   IOPT=0 does the FFT(s) directly since tables exist.

   IOPT<0 creates requested tables and returns.

   If IOPT is negative, the trigonometric tables TA and/or TB will be created, but nothing else takes place. To actually get an FFT, IOPT must be zero or positive.

2. Its absolute value determines which, if any, of the trigonometric tables TA and TB should be created by this routine:

   ABS(IOPT)=0 indicates TA and (if needed) TB exist.

   ABS(IOPT)=1 indicates TA should be computed.

   ABS(IOPT)=2 indicates TB should be computed if needed.

   ABS(IOPT)=3 combines ABS(IOPT)=1 and ABS(IOPT)=2.

IOPT=-2,+2,-3, or +3 does not guarantee that TB is created; this happens only if this routine plans to use it. The decision is based on the values of L, N, and MODE, as well as on that of IOPT.

Once created, these tables are never modified and can be used in subsequent calls.

When IOPT is negative, the parameters X, DTYP, S, and XNORM are ignored.

Parameters (Continued)

MODE

This parameter must be of type integer and has two functions:

1. Its sign determines the type of storage used. A positive value indicates that array X contains the data stored with real and imaginary parts separated, as described under X above. A negative value indicates that the normal FORTRAN storage mode is used; that is, that the real and imaginary parts are interleaved. The scheme with separated parts is optimal for this routine. If MODE is negative, a certain penalty is incurred since the routine in this case will begin by separating the real and imaginary parts and end by interleaving them again.

2. Its absolute value specifies whether the expanded trigonometric table TB should be used or not. MODE=+1 or -1 indicates that the space TB is available and can be used if deemed advantageous by this routine. MODE=+2 or -2 prohibits the use of TB. Substantial savings can be realized by allowing for the use of TB (see Method).

Externals

SCRRI, SCRRF, FAC2U, FAC4, SIN, COS

Dynamic Space

1024 words.

Remarks

The combination of the DTYP and MODE parameter specifications define four specific cases:

CASE 1. DTYP='CD' and MODE>0.

This is the standard case: complex vectors in and complex vectors out with real and imaginary parts separated.

INPUT: REAL X(0:L-1,0:2*N-1), an array with column length L. Each of the L rows should hold one complex vector with N complex elements. Its real parts should occupy the first N and its imaginary parts the last N locations of that row. Thus, the real parts of the Jth vector, (J=0,1,...,L-1), should occupy locations (X(J,K) , K=0,N-1) and the imaginary parts locations (X(J,N+K) , K=0,N-1).

OUTPUT: Array X contains the result of the forward (XNORM positive) or inverse (XNORM negative) Fourier transform of the L input vectors, stored in the same fashion as was the input data.

CASE 2. DTYP='CD' and MODE<0.

This differs from case 1 in that the normal FORTRAN storage method for complex elements is used. Thus, real and imaginary parts are interleaved. Note that there is some penalty associated with using this type of storage. This routine starts by converting the data to the form with separated real and imaginary parts (case 1) and ends with the reversal of that process.

INPUT: COMPLEX X(0:L-1,0:N-1), an array with column length L, as measured in units of complex elements. Each of the L rows should hold one complex vector of length N. Thus, the Kth complex element, (K=0,1,...,N-1), of the Jth vector, (J=0,1,...,L-1), occupies the complex element (double word) location X(J,K).

OUTPUT: Array X contains the result of the forward (XNORM positive) or inverse (XNORM negative) Fourier transform of the L input vectors, stored in the same fashion as was the input data.

Remarks (Continued)

CASE 3. DTYP='RD' and MODE>0.

This requests the forward FFT (XNORM positive) of the 2L real vectors of length N into 2L conjugate symmetric complex vectors of length N, or the reverse operation (XNORM negative). Note that whenever DTYP='RD', it is necessary to use only even values for N.

When a Fourier transform is applied to a real vector with N elements the result is a complex vector with N elements. Thus, N real values are transformed into 2N real (=N complex) values, implying a redundancy in the result. Indeed, the transformed vector is conjugate symmetric around its midpoint, so that N values trivially can be derived from the other N.

Mathematically this can be expressed as follows:

When the real vector (x(j) , j=0,...,n-1) is Fourier transformed, the resulting complex vector (z(k) , k=0,...,n-1) has the property that (z(n-k)=conjugate(z(k)) , k=0,...,n-1).

Using zr and zi to denote real and imaginary parts, respectively, and restricting ourselves to even-valued n's, we may reformulate this as follows:

```
zi(0)   =  0
zi(n/2) =  0
zr(n-k) =  zr(k)      k=1,2,...,n/2-1
zi(n-k) = -zi(k)      k=1,2,...,n/2-1
```

Thus, the vector z may be completely defined by specifying the n/2+1 real parts of (z(k) , k=0,...,n/2) and the n/2-1 imaginary parts of (z(k) , k=1,...,n/2-1). This amounts to a total of n real values. The point around which z exhibits conjugate symmetry is z(n/2), often referred to as the Nyquist frequency component, in spite of the fact that far from all Fourier transforms correspond to a change in variable from time to frequency. z(0) is often referred to as the DC frequency component.

Real storage mode (input when XNORM>0, output when XNORM<0):

REAL X(0:L-1,0:2*N-1), the same as when DTYP='CD'. However, in this case we distinguish between 2L real vectors. Each of the L rows holds two real vectors, each of length N: the Jth row, (J=0,1,...,L-1), contains one real vector in locations (X(J,K) , K=0,...,N-1) and another one in (X(J,N+K) , K=0,...,N-1).

Complex storage mode (output when XNORM>0, input when XNORM<0):

N/2+1 real and N/2-1 imaginary parts of 2L conjugate symmetric complex vectors, each of length N. In the real array X(0:L-1,0:2*N-1), the Jth row, (J=0,1,...,L-1), contains two complex vectors. The first one has its first N/2+1 real parts in locations (X(J,K) , K=0,...,N/2) and the imaginary parts that correspond to elements (1,2,...,N/2-1) in locations (X(J,N/2+K) , K=1,...,N/2-1). Similarly, the second vector in the Jth row has its first N/2+1 real parts in locations (X(J,N+K) , K=0,...,N/2) and its imaginary parts corresponding to elements (1,2,...,N/2-1) in locations (X(J,3N/2+K) , K=1,...,N/2-1).

CASE 4. DTYP='RD' and MODE<0.

This case differs from the previous case only in that the transformed data, which consists of complex numbers, is stored in the standard FORTRAN manner; that is, with the real and imaginary parts interleaved. As was the case in CASE 2 above, some extra overhead is incurred in order to make this come about. The Fourier transforms themselves are always performed with separated real and imaginary parts, so that the interleaving takes place after the real-to-complex FFT, while the act of separating precedes the complex-to-real FFT.

Remarks (Continued)

Real storage mode (input when XNORM>0, output when XNORM<0):

REAL X(0:L-1,0:2*N-1), the same as when DTYP='CD'. However, in this case we distinguish between 2L real vectors. Each of the L rows holds two real vectors, each of length N: the Jth row, (J=0,1,...,L-1), contains one real vector in locations (X(J,K) ,K=0,...,N-1) and another one in (X(J,N+K) , K=0,...,N-1).

Complex storage mode (output when XNORM>0, input when XNORM<0):

COMPLEX X(0:L-1,0:N-1), where each of the L rows contains N/2+1 real and N/2-1 imaginary parts of two conjugate symmetric complex vectors, each of length N. In a given row J, where J=0,1,...,L-1, there are N complex numbers. The first N/2 of these completely define one conjugate symmetric complex vector, and the same is true for the latter N/2 of them. The first element, X(J,0), is a representation of two complex elements for which we know that the imaginary parts are zero: the DC and the Nyquist component. Thus, the DC component (the first element) of the transformed vector is actually CMPLX(REAL(X(J,0)) , 0.0), while the Nyquist component can be formed as the complex number CMPLX(IMAG(X(J,0)) , 0.0). Elements with subscripts 1 through N/2-1 are stored in the complex array locations (X(J,K),K=1,...,N/2-1), while elements with subscripts N/2+1 through N-1 can be derived by taking the complex conjugates of (X(J,N-K),K=N/2+1,...,N-1). Similarly, the DC and Nyquist components of the second vector in row J are CMPLX(REAL(X(J,N/2)),0.0) and CMPLX(IMAG(X(J,N/2)),0.0), respectively. Elements with subscripts 1 through N/2-1 are stored in locations (X(J,N/2+K),K=1,...,N/2-1), while elements with subscripts N/2+1 through N-1 can be derived by taking the complex conjugates of (X(J,N/2+N-K),K=N/2+1,...,N-1).

Method

Given a vector (x(k) , k=0,1,2,...,n-1), its Fourier transform (y(j) , j=0,1,2,...,n-1) is defined by

$$y(j) = \sum_{k=0}^{n-1} x(k)*e^{-2\pi ijk/n}$$

where $i^2=-1$. The inverse transform is obtained by changing -2π to $+2\pi$.

N is assumed to be the product of p factors of 2, q factors of 3, and r factors of 5. The following processing phases can be identified:

NOTE

Currently q=r=0 is required.

1. If DTYP='RD' and XNORM is negative, the whole data volume must be slightly manipulated before the actual inverse Fourier transform can take place.

2. If MODE=+1 or -1 the routine processes m factors of 2, ($m \leq p$), one at a time, using a variation of Urich's method. m is determined internally so as to obtain the smallest possible execution time. When L is large, m is zero, while for small values of L, m may be as large as 6.

   Significant savings can be realized by allowing for this method. The value of L above for which this is not true varies with N, but will mostly fall between 8 and 16. Thus, for smaller values of L it pays to use N-values containing many factors of 2.

Method (Continued)

3. The remaining factors of 5 are now processed by applying a standard base-5 Sande-Tukey algorithm to the data. This is followed by the processing of the remaining factors of 4, and then those of 3. If after this a single factor of 2 remains, this is handled as the last factor.

4. If DTYP='RD' and XNORM is positive, the data must now be manipulated somewhat before return.

CYBER 990 timing information

The tables below show measured execution times for different values of L and N when FFT1D was executed on a dedicated CYBER 990. In all cases IOPT=0. For L values below 16, MODE=1 was used, while MODE=2 was the choice for L=16 and larger.

Microseconds per complex transform (time/L when DTYP='CD')

	N=:	32	64	128	256	512	1024	2048	4096
L= 1	MODE=1	210	300	480	820	1500	3000	6000	11700
L= 8	MODE=1	63	120	250	520	1000	2100	4900	10600
L=16	MODE=2	39	77	170	360	790	1700	3700	9700
L=32	MODE=2	26	55	130	270	610	1300	3400	6600
L=64	MODE=2	20	45	110	230	540	1200	2900	6100

Microseconds per real transform (time/2L when DTYP='RD')

	N=:	32	64	128	256	512	1024	2048	4096
L= 1	MODE=1	120	160	250	430	810	1600	3100	6100
L= 8	MODE=1	36	67	140	280	550	1100	2600	5600
L=16	MODE=2	22	43	96	200	440	920	2000	5300
L=32	MODE=2	15	31	71	150	350	760	1900	3800
L=64	MODE=2	12	26	59	130	300	660	1500	3100

Subroutine F64TR4

Purpose

To convert one or several Control Data 64-bit floating-point numbers to IBM 32-bit floating-point format, also known as REAL*4.

Format

CALL F64TR4 (N , A , B , IXBYTB)

Parameters

N

Number of Control Data 64-bit floating-point numbers to convert. Must be of type integer.

A

Input array of type real, holding N contiguously stored Control Data 64-bit floating-point numbers. The contents of A are not altered by this routine, provided A and B are not the same array.

B

Ouput array in which the converted numbers are stored contiguously. Can be of any type. Each 4-byte result is stored in byte natural order. Conversion in place is allowed; that is, B may represent the same address as A. See Remarks below for details about what happens when a Control Data number in A is an indefinite, an infinity, or simply lies outside the range representable in the IBM 32-bit floating-point format.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B where the first converted number should be placed. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Remarks

The IBM 32-bit floating-point format features a sign bit (0=plus, 1=minus) to the far left, followed by a 7-bit exponent expressed in base 16 and biased by 64 (making it positive). The rightmost 24 bits constitute a normalized, positive coefficient. The normalization is done 4 bits at a time, so that the most significant 4 bits of the coefficient may form any hexadecimal digit except 0.

Since (for positive numbers) the 1's and 2's complement notations are equivalent, no such distinction is made above.

There are some special cases of the IBM 32-bit floating-point format that may result from a conversion of a Control Data 64-bit floating-point number:

1. When the Control Data number is an indefinite; that is, when the 4 leftmost bits make up either one of the two hexadecimal digits "7" (positive indefinite) or "F" (negative indefinite), then this number is converted to the "reserved" IBM quantity with the hexadecimal representation Z"7FFFFFFF".

2. When the absolute value of the Control Data number equals infinity; that is, when the 4 leftmost bits make up either one of the hexadecimal digits "5", "6", "D", or "E", then this number is also converted to Z"7FFFFFFF".

3. When the absolute value of the Control Data number is too large for the IBM representation; that is, when the computed IBM biased exponent exceeds 255, then this number is also converted to Z"7FFFFFFF".

4. When the absolute value of the Control Data number is too small for the IBM representation; that is, when the computed IBM biased exponent is negative, then a zero is generated (32 zero bits).

Subroutine F64TSR

Purpose

To convert one or several Control Data 64-bit floating-point numbers to IEEE 32-bit floating-point format, also known as Short Real. Each resulting 4-byte number is generated in either natural (F64TSRN) or byte reversed (F64TSR) order.

Format

```
CALL F64TSR ( N , A , B , IXBYTB )
CALL F64TSRN( N , A , B , IXBYTB )
```

Parameters

N

Number of Control Data 64-bit floating-point numbers to convert. Must be of type integer.

A

Input array of type real, holding N contiguously stored Control Data 64-bit floating-point numbers. The contents of A are not altered by this routine, provided A and B are not the same array.

B

Output array in which the converted numbers are stored contiguously. Can be of any data type. Each 4-byte result is stored in natural order (0123) when F64TSRN is called, and in byte reversed order (3210) when F64TSR is called. Conversion in place is allowed; that is, B may represent the same address as A. See REMARKS below for details about what happens when a Control Data number in A is an indefinite, an infinity, or simply lies outside the range representable in the IEEE Short Real format.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B where the first converted number should be placed. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Remarks

The IEEE Short Real, or 32-bit floating-point format is a sign/magnitude format, featuring a sign bit (0=plus, 1=minus), an 8-bit base 2 exponent biased by decimal 127, and a 24-bit positive coefficient that in general is normalized. Since the coefficient of a positive normalized number always starts with a 1, the first bit is redundant. The bit is indeed omitted from the actual floating-point representation. It is referred to as "the hidden bit". There is an implied binary point between the two most significant bits of the coefficient; that is, just after the hidden bit. Since for positive numbers the 1's and 2's complement notations are equivalent, no such distinction is made above.

There are some special cases of the IEEE Short Real format that may result from a conversion of a Control Data 64-bit floating-point number:

1. When the Control Data number is an indefinite; that is, when the 4 leftmost bits make up either one of the two hexadecimal digits "7" (positive indefinite) or "F" (negative indefinite), then this number is converted to the IEEE quantity NaN (Not-a-Number), characterized by a sign, a biased exponent value of 255, and a nonzero coefficient (fraction). For this purpose this routine arbitrarily chooses the 23-bit hexadecimal coefficient "700000". Thus, the positive indefinites get translated into Z"7FF00000", and the negative indefinites into Z"FFF00000".

2. When the absolute value of the Control Data number equals infinity; that is, when the 4 leftmost bits make up either one of the hexadecimal digits "5", "6", "D", or "E", then this number is converted into an IEEE signed infinity, characterized by a sign, a biased exponent value of 255, and a zero-valued coefficient (fraction). Thus, the two possible outcomes are Z"7F800000" (positive) and Z"FF800000" (negative).

3. When the absolute value of the Control Data number is too large for the IEEE representation; that is, when the computed IEEE biased exponent equals or exceeds 255, then this number is converted into a signed infinity. See item 2 above.

4. When the absolute value of the Control Data number is too small for the IEEE representation; that is, when the computed IEEE biased exponent equals zero or is negative, then a conversion to an unnormalized number is attempted. If the absolute value is too small for this as well, then a zero is generated (32 zero bits). An unnormalized IEEE number is characterized by a sign, a biased exponent value of zero, and a nonzero coefficient. In this case the bias is assumed to be decimal 126 (rather than 127), and the hidden bit is assumed to equal 0 (rather than 1).

Subroutine F64TVF

Purpose

To convert one or several Control Data 64-bit floating-point numbers to VAX 32-bit floating-point format.

Format

```
CALL F64TVF( N , A , B , IXBYTB )
```

Parameters

N

Number of Control Data 64-bit floating-point numbers to convert. Must be of type integer.

A

Input array of type real, holding N contiguously stored Control Data 64-bit floating-point numbers. The contents of A are not altered by this routine, provided A and B are not the same array.

B

Output array in which the converted numbers are stored contiguously. Can be of any data type. Each 4-byte result is stored in pairwise byte reversed order (1032 instead of 0123). Conversion in place is allowed; that is, B may represent the same address as A. See Remarks below for details about what happens when a Control Data number in A is an indefinite, an infinity, or simply lies outside the range representable in the VAX 32-bit floating-point format.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B where the first converted number should be placed. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Remarks

The VAX 32-bit floating-point format is a sign/magnitude format, featuring a sign bit (0=plus, 1=minus), an 8-bit base 2 exponent biased by decimal 128, and a 24-bit normalized positive coefficient. Since the coefficient of a positive normalized number always starts with a 1, the first bit is redundant. The bit is indeed omitted from the actual floating-point representation. It is referred to as "the hidden bit". There is an implied binary point to the left of the hidden bit.

Since for positive numbers the 1's and 2's complement notations are equivalent, no such distinction is made above.

There are some special cases of the VAX 32-bit floating-point format that from a conversion of a Control Data 64-bit floating-point number:

1. When the Control Data number is an indefinite; that is, when the 4 leftmost bits make up either one of the two hexadecimal digits "7" (positive indefinite) or "F" (negative indefinite), then this number is converted to the "reserved" VAX quantity with the hexadecimal representation Z"80000000".

2. When the absolute value of the Control Data number equals infinity; that is, when the 4 leftmost bits make up either one of the hexadecimal digits "5", "6", "D", or "E", then this number is also converted to Z"80000000".

3. When the absolute value of the Control Data number is too large for the VAX representation; that is, when the computed VAX biased exponent exceeds 255, then this number is also converted to Z"80000000".

4. When the absolute value of the Control Data number is too small for the VAX representation; that is, when the computed VAX biased exponent equals zero or is negative, then a zero is generated (32 zero bits).

Subroutine GENSPD

Purpose

To create the upper triangular part of a symmetric positive definite matrix using the system-provided random number generator RANF.

Format

CALL GENSPD (N , S , KS , SCALE)

Parameters

N

The order of the source matrix S. Must be of type integer.

S

Output array into which the upper triangular part of the symmetric positive definite matrix is written. Must be of type real.

KS

Storage mode indicator for S. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of S are stored linearly, thus omitting the subdiagonal elements:

$$s_{00}-s_{01}-s_{11}-s_{02}-s_{12}-s_{22}-s_{03}-\cdots$$

A value greater than N indicates full storage mode; that is, KS is the size of the first dimension (column length) of the two-dimensional array S. In this case the subdiagonal elements are ignored; only the N(N+1)/2 elements of the upper triangular part are written.

SCALE

A scale factor of type real which is applied to the elements of S, as outlined below under Method.

Externals

SQRT, RANF, VRANF, RVSUB, RVMUL

Dynamic Space

None

Method

The N(N-1)/2 superdiagonal elements of the matrix S are assigned randomly generated values in the range [-SCALE,SCALE]. The N diagonal elements are assigned randomly generated values in the range SCALE * SQRT(N) * [1.0,2.0]. The result is that the upper triangular part (N(N+1)/2 elements) of a symmetric positive definite matrix is generated.

ITPACKV 2C - Iterative Solvers

Purpose

To solve a large sparse linear system of equations of the form AX=B, where A is a symmetric and positive definite (or slightly nonsymmetric) [N,N] matrix, B is the right-hand side, and X is the desired solution vector. The routines can handle more nonsymmetric systems as long as the diagonal elements of the coefficient matrix are positive, but the convergence characteristics will not be as good. All routines use some adaptive iterative algorithm, as specified under Format.

Format

```
CALL rtn ( N , NDIM , MAXNZ , JCOEF , COEF , RHS , X , IWKSP , NW , WKSP ,
           IP , RP , IER )
```

Where rtn is one of the keywords in the following table:

Keyword	Routine Selected
JCG	Jacobi with CG acceleration
JSI	Jacobi with SI acceleration
SOR	Successive overrelaxation
SSORCG	Symmetric SOR with CG acceleration
SSORSI	Symmetric SOR with SI acceleration
RSCG	Reduced system with CG acceleration
RSSI	Reduced system with SI acceleration

Three of the iterative procedures used are available with either one of two acceleration procedures for rapid convergence: conjugate gradient, CG, and Chebyshev or semi-iteration, SI.

Parameters

N

The order of the linear system. Must be of type integer.

NDIM

Size of the first dimension (column length) of each of the two arrays COEF and JCOEF. Must be of type integer. Must be at least as large as N.

MAXNZ

The maximum number of nonzero entries in any row of the coefficient matrix A. Must be of type integer.

JCOEF

Input array of dimension (NDIM,MAXNZ). Must be of type integer. This is an index array which must contain a user-created map between corresponding locations in the array JCOEF and the matrix A: the matrix element A(J,JCOEF(J,K)) should be stored at location COEF(J,K). Expressed differently: the array element COEF(J,K) actually belongs in row J and column JCOEF(J,K) of the coefficient matrix A. Unused entries in JCOEF must be set to zero. The contents of JCOEF will in general not be preserved. See Examples.

COEF

Input array dimensioned (NDIM,MAXNZ). Must be of type real. This array must on input contain all nonzero entries of the matrix A, located in positions determined by the contents of array JCOEF (see description above). Unused locations must be set to zero. The contents of COEF will in general not be preserved. See Examples.

Parameters (Continued)

RHS

Input array of length at least N, holding the single right-hand side vector to be solved for. Must be of type real. RHS will be internally scaled, and then unscaled before exit. Thus its contents will be slightly altered by the introduction of rounding errors.

X

Array of length at least N, on input holding the initial guess (use zeroes if not known), and on output the final approximate solution. Must be of type real.

IWKSP

Array of length at least 3N words, used as work space. Must be of type integer. When reindexing for red-black ordering, the first N locations contain on output the permutation vector for the red-black indexing, and the next N locations contain the inverse of this vector.

NW

Available length of array WKSP. Must be of type integer. The actual amount used may on output be found in IP(8). The minimum value of NW is as follows:

Routine	Minimum Value
JCG	5*N + K*ITMAX
JSI	3*N
SOR	2*N
SSORCG	7*N + K*ITMAX
SSORSI	6*N
RSCG	2*N + 3*NBLACK + K*ITMAX
RSSI	2*N + NBLACK

where K = 2 for symmetric problems.
= 4 for nonsymmetric problems.
ITMAX = max number of iterations allowed (=IP(1)).
NBLACK = number of black grid points in the red-black ordering (=IP(9)).

WKSP

Array of length at least NW, used as work space. Can be of any type except character.

IP

Input/output array of length at least 12 words. Must be of type integer. The user must supply values in the first 12 locations, but on output some will have been changed as detailed below. In the following description each location is given a name, and a default value is specified as [NAME=VALUE]. The default values for arrays IP and RP may be collectively assigned by means of a call to the ITPACK subroutine DFAULT: CALL DFAULT(IP,RP). No other automatic mechanism for assigning default values exists.

Parameters (Continued)

IP(1) [ITMAX=100] Maximum number of iterations allowed. Will be reset to the actual number of iterations performed on output.

IP(2) [LEVEL=0] Output level control switch. Controls the amount of output written to the file with the unit number given in IP(4). The higher the value, the more output:

- < 0 No output
- $= 0$ Fatal error messages only
- $= 1$ Warning messages and minimum output
- $= 2$ Reasonable summary
- $= 3$ Parameter values and informative comments
- $= 4$ Approximate solution after each iteration
- $= 5$ The full original system

IP(3) [IRESET=0] Communication switch.

- $= 0$ Implies values in IP and RP will be overwritten as indicated in the respective descriptions.
- $\neq 0$ Only IP(1) and IP(8) will be reset.

IP(4) [NOUT=6] Logical unit number for printed output.

IP(5) [ISYM=0] Symmetric matrix switch.

- $= 0$ Matrix is symmetric
- $= 1$ Matrix is nonsymmetric

IP(6) [IADAPT=1] Adaptive strategy switch. Determines which, if any, of the parameters SME, CME, OMEGA, SPECR and BETAB should remain fixed, and which should be computed automatically in either a fully or partially adaptive sense. See also the descriptions below of RP(K), K=2,3,5,6,7.

- $= 0$ Nonadaptive. Fixed values used for all parameters.
- $= 1$ Fully adaptive procedures used for all parameters.
- $= 2$ (SSOR methods only) SPECR determined adaptively. Remaining parameters are kept fixed.
- $= 3$ (SSOR methods only) BETAB is kept fixed. Remaining parameters are determined adaptively.

IP(7) [ICASE=1] Adaptive procedure case switch--meaningful only for JSI and SSOR methods. There are two strategies, called Case i and ii, for doing the adaptive procedure. The selection is based on knowledge about the eigenvalues of the Jacobi matrix derived from the coefficient matrix A:

- $\neq 2$ Case i: SME is kept fixed at the initial value, unless it is zero, in which case it will be changed to minus one. If the user specifies a value for SME that differs from zero, care should be taken so that it does not exceed the minimum eigenvalue of the Jacobi matrix.
- $= 2$ Case ii: used when it is known that the absolute value of the minimum eigenvalue does not exceed the value of the maximum eigenvalue. Here SME is always set to -CME, which may change adaptively.

Parameters (Continued)

The case switch determines how the estimates SME (=RP(3)) and CME (=RP(2)) are recomputed adaptively. As far as the adaptive procedure is concerned, Case i is the most general, and should be specified in the absence of specific knowledge about the eigenvalues. An example when Case ii is appropriate occurs when the Jacobi matrix is 2-cyclic; that is, when it is either diagonal or can be reordered into a red-black matrix, since then the smallest eigenvalue equals the negative of the largest. Also, if A is an L-matrix (positive diagonal, nonpositive off-diagonal elements), the condition for Case ii is satisfied. Selecting the correct case may increase the rate of convergence of the algorithm.

IP(8) [NWKSP=0] Amount of workspace used; that is, required length of array WKSP. Used for output only.

IP(9) [NBLACK=-2] Red-black ordering switch. On output, if reindexing is done, NBLACK is set to the order of the black subsystem. On input, the meaning is:

- = -2 Skip indexing: system already in desired form and is not red-black.
- = -1 Compute red-black indexing and permute the system.
- >= 0 Skip indexing: system already in red-black form, with NBLACK as the order of the black subsystem.

IP(10) [IREMOVE=0] Switch for removing rows and columns when the diagonal entry is extremely large compared to the nonzero off-diagonal entries in that row. See also RP(8).

- = 0 Not done.
- ≠ 0 Test and removal performed.

IP(11) [ITIME=0] Timing switch.

- = 0 Measure execution time.
- ≠ 0 Do not measure the execution time.

IP(12) [IDGTS=0] Error analysis switch, determining what type of error analysis of the last computed solution should be done; that is, how the accuracy of the solution should be communicated back to the caller.

- < 0 Skip error analysis and do not print anything.
- = 0 Compute RP(11)=DIGIT1 and RP(12)=DIGIT2.
- = 1 Print DIGIT1 and DIGIT2.
- = 2 Print final approximate solution vector.
- = 3 Print final approximate residual vector.
- = 4 Print both solution and residual vectors.

Parameters (Continued)

RP

Input/output array of length at least 12. Must be of type real. The user must supply values in the first 12 locations, but on output some will have been changed as detailed below. In the following description each location is given a name, and a default value is specified as [NAME=VALUE]. The default values for arrays IP and RP may be collectively assigned by means of a call to the ITPACK subroutine DFAULT: CALL DFAULT(IP,RP). No other automatic mechanism for assigning default values exists.

RP(1) [ZETA=.000005] Stopping criterion, or approximate relative accuracy requested in the final solution. If the method does not converge in IP(1)=ITMAX iterations, then RP(1) is reset to an estimate of the relative accuracy achieved. The stopping criterion is a test of whether ZETA is greater than the ratio of the Euclidean norm of the pseudo-residual vector and the norm of the current iteration vector times a constant involving an eigenvalue estimate. The pseudo-residual vector is G*X(N) + K - X(N), where the basic iterative method is of the form X(N+1) = G*X(N) + K.

RP(2) [CME=0.0] Estimate of the largest eigenvalue of the Jacobi matrix. Changes to new estimate if adaptive procedure is used.

RP(3) [SME=0.0] Estimate of the smallest eigenvalue of the Jacobi matrix for the JSI method. In Case i (see IP(7)) SME is fixed throughout at a value not exceeding the minimum eigenvalue of the Jacobi matrix. In Case ii SME is always set to -CME, with CME changing in the adaptive procedure.

RP(4) [FF=0.75] Adaptive procedure damping factor. Should be a positive number not exceeding 1.0. FF=1.0 causes the most frequent parameter changes when the fully adaptive switch IP(6) equals 1.

RP(5) [OMEGA=1.0] Overrelaxation parameter for SOR and SSOR methods. OMEGA changes if the method is fully adaptive.

RP(6) [SPECR=0.0] Estimate of the spectral radius of the SSOR matrix. SPECR changes if the method is adaptive.

RP(7) [BETAB=0.25] Estimate of the spectral radius of the LU matrix used in SSOR methods. BETAB may change depending on the value of the adaptive switch IP(6). The matrix L is the strictly lower triangular part of the Jacobi matrix and U is the strictly upper triangular part. When the spectral radius of LU is less than or equal to 0.25 the "SSOR condition" is satisfied for some problems, provided the natural ordering is used.

RP(8) [TOL=100.*SRELPR] Tolerance factor, near the machine relative precision SRELPR. The value of SRELPR is set internally to HEX(3FD1 8000 0000 0000) or 2**-48 (approximately 3.55 E-15). In each row, if all nonzero off-diagonal entries are smaller in magnitude than the diagonal entry, this row and the corresponding column are essentially removed from the system. This is done by setting the off-diagonal elements in the row and column to zero, and replacing the diagonal element with 1.0. An adjustment of the elements on the right-hand side is also done, so that the new system is equivalent to the old one. If the diagonal element is the only nonzero element in a row and is not greater than the reciprocal of TOL, then no elimination is done. This procedure is useful for linear systems arising from finite-element discretizations of partial differential equations in which Dirichlet boundary conditions are handled by making the diagonal elements extremely large.

Parameters (Continued)

RP(9) [TIME1=0.0] Total time in seconds from beginning of the iterative algorithm until convergence.

RP(10) [TIME2=0.0] Total time in seconds for the entire call.

RP(11) [DIGIT1=0.0] Approximate number of digits of accuracy using the estimated relative error with the final approximate solution. Computed as the negative of the base-10 logarithm of the final value of the stopping test.

RP(12) [DIGIT2=0.0] Approximate number of digits of accuracy using the estimated relative residual with the final approximate solution after the algorithm has converged. Computed as the negative of the base-10 logarithm of the ratio of the 2-norm of the residual vector and the 2-norm of the right-hand side vector. This estimate is related to the condition number of the original system, and will therefore not be accurate if the system is ill-conditoned. If DIGIT2 differs significantly from DIGIT1, then either the stopping test has not worked successfully or the original system is ill-conditioned.

IER

Error return parameter, set by the called ITPACK routine. Must be of type integer. A value of 0 indicates normal convergence. Other values are of the form M+J, where M=10,20,30,40,50,60,70 indicates which of the main solution modules was called, and M=400,500,600 points to the routines SCAL, ZBRENT and EQRT1S, respectively, as the routine where the error was detected (SCAL scales the system, ZBRENT and EQRT1S are used for eigenvalue estimation in the symmetric and nonsymmetric cases, respectively).

IER = M + J <= 74:

where:	M = 10	JCG module called
	M = 20	JSI module called
	M = 30	SOR module called
	M = 40	SSORCG module called
	M = 50	SSORSI module called
	M = 60	RSCG module called
	M = 70	RSSI module called
	J = 1	Invalid order of the linear system.
	J = 2	Insufficient amount of workspace assigned. Check IP(8) for the required amount.
	J = 3	Failure to converge after ITMAX iterations. Check RP(1) for last computed stopping value.
	J = 4	Invalid order of black subsystem with red-black indexing.
	IER = 401	Zero diagonal element.
	IER = 402	Nonexistent diagonal element.
	IER = 501 502 601	Difficulty encountered in eigenvalue estimation.
	IER = 602	Matrix is not positive definite.

Externals

1. Various LIB99 routines, performing basic vector arithmetic: RVAXPY, RSMULV, RVADD, and others.

2. Routines from the set of 46 internal ITPACK subroutines and functions. Each of the seven user-callable routines mentioned under CALLS performs its work by means of direct or indirect calls to some of these internal routines, none of which are directly user-callable and therfore not documented. Below is a list of these internal entry-points:

CHEBY	CHGCON	CHGSI	CHGSME	DETERM	DFAULT	ECHALL	EIGVNS
EIGVSS	EQRT1S	ITJCG	ITJSI	ITSOR	ITSRCG	ITSRSI	ITRSCG
ITRSSI	IPSTR	ITERM	MOVE	OMEG	OMGCHG	OMGSTR	PARCON
PARSI	PBETA	PBSOR	PERMAT	PERROR	PERVEC	PFSOR	PFSOR1
PJAC	PMULT	PRBNDX	PRSBLK	PRSRED	PSTOP	PVTBV	SBELM
SCAL	TAU	TSTCHG	UNSCAL	VOUT	ZBRENT		

3. Three common blocks are used to communicate 32 scalar variables: /ITCOM1/, /ITCOM2/ and /ITCOM3/.

Dynamic Space

None

Method

The basic solution methods provided are the Jacobi, SOR, symmetric SOR, and reduced system methods. All methods except SOR are accelerated by either conjugate gradient or Chebyshev acceleration. When using the reduced system methods it is required that the system be reordered into a red-black system; that is, a system of the form

$$\begin{bmatrix} D1 & K \\ H & D2 \end{bmatrix}$$

Where D1 and D2 are diagonal matrices. See also the description of the calling parameter IP(9) above.

Each of the solution modules scales the linear system to a unit diagonal system prior to iterating, and unscales it upon termination. This reduces the number of arithmetic operations, but it may introduce small changes in the coefficients (COEF) and the right-hand side vector (RHS) due to rounding errors.

When requested, a red-black permutation of the linear system will be done before and after the iterative algorithm is started. If not, the linear system is used in the order it is provided by the caller. If SOR, SSORCG, OR SSORSI is called, an attempt will be made to segregate the parts of COEF and JCOEF corresponding to the upper and lower triangular parts of matrix A into separate columns. Since this will in general take more storage, it may not be possible if MAXNZ is too small. In this case the algorithms will operate on the existing data structure, but performance will be degraded. Segregation of L and U increases the vectorizability of these algorithms. In the case that red-black ordering is used with these routines, no attempt to segregate L and U is made.

The successive overrelaxation (SOR) method has been shown to be more effective with the red-black ordering than with the natural ordering for some problems. In the SOR algorithm the first iteration uses OMEGA=1.0, and the stopping criterion is set to a large value so that at least one Gauss-Seidel iteration is performed before an approximate value for the optimum relaxation parameters is computed.

Example

The sparse coefficient matrix A of dimension (N,N) is expected in a compressed format in array COEF, dimensioned (NDIM,MAXNZ). A map between A and COEF must be provided in array JCOEF, which should be dimensioned the same way as COEF. The structure of this storage is detailed in the parameter description of COEF and JCOEF above, but we will here illustrate the scheme with an example and then make some important remarks.

Let the symmetric 5 by 5 matrix A be defined as follows

$$A = \begin{bmatrix} a11 & 0.0 & 0.0 & a14 & a15 \\ 0.0 & a22 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & a33 & 0.0 & 0.0 \\ a14 & 0.0 & 0.0 & a44 & a45 \\ a15 & 0.0 & 0.0 & a45 & a55 \end{bmatrix}$$

The contents of COEF and JCOEF should then be:

$$COEF = \begin{bmatrix} a11 & a14 & a15 \\ a22 & 0.0 & 0.0 \\ a33 & 0.0 & 0.0 \\ a44 & a14 & a45 \\ a55 & a15 & a45 \end{bmatrix} \qquad JCOEF = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 1 & 5 \\ 5 & 1 & 4 \end{bmatrix}$$

1. Unused locations in COEF and JCOEF must be filled with zeroes.

2. The nonzero elements in a given row of COEF may appear in any order. However, if the diagonal element is not in column 1, then the called ITPACK module will place it there without returning it to its original position upon exiting.

3. The diagonal element of each row should be positive. If a diagonal element is negative, the called ITPACK module will reverse the sign of all entries corresponding to this equation. This may result in a loss of symmetry of the system, and the convergence may be adversely affected.

4. All nonzero matrix entries must be present, even those duplicated due to the symmetry of the system.

Remarks

The LIB99 version of the iterative package ITPACK is for all practical purposes identical to ITPACKV 2C, as obtained directly from the University of Texas. To achieve satisfactory vector performance many DO loops were replaced by calls to basic vector arithmetic LIB99 routines, such as RVAXPY, RSMULV, RVADD, and others. However, this does in no way alter the algorithmic formulation.

The iterative algorithms used in ITPACK are quite complicated, and some knowledge of iterative methods is necessary in order to completely understand them. The interested reader can use the comment sheet in the back of this manual to request additional information.

Subroutine I64TSI

Purpose

To convert one or several Control Data 64-bit integers to IEEE 16-bit integer format, also known as Short Integer format (INTEGER*2 in IBM notation). Each resulting 2-byte number is generated in either natural (I64TI2) or byte reversed (I64TSI) order.

Format

```
CALL I64TI2( N , A , B , IXBYTB )
CALL I64TSI( N , A , B , IXBYTB )
```

Parameters

N

Number of Control Data 64-bit integers to convert. Must be of type integer.

A

Input array of type integer, holding N contiguously stored Control Data 64-bit integers. The contents of A are not altered by this routine, provided A and B are not the same array.

B

Output array in which the converted numbers are stored contiguously. Can be of any data type. Each 2-byte result is stored in natural order (01) when I64TI2 is called, and in byte reversed order (10) when I64TSI is called. Conversion in place is allowed; that is, B may represent the same address as A. See Remarks below for details about what happens when the value of a Control Data number in A lies outside the range representable in the IEEE Short Integer format.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B where the first converted number should be placed. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Remarks

An IEEE Short Integer is a 16-bit 2's complement integer. The range of numbers representable in this format is [-32768,32767], inclusive. However, some hardware reserves the most negative number (hexadecimal Z"8000") for some special purpose, not recognizing it as an integer.

This routine converts integers by simply isolating the lower 16 bits of the CDC 64-bit integers furnished in array A. Thus, integers outside the range specified above will be converted, but their value will change. For instance, the two integers 43981 and -21555 (hexadecimal Z"000000000000ABCD" and Z"FFFFFFFFFFFFABCD", respectively) are both converted to the Short Integer Z"ABCD", with a decimal value of -21555.

Subroutine I64TWI

Purpose

To convert one or several Control Data 64-bit integers to IEEE 32-bit integer format, also known as Word Integer format (INTEGER*4 in IBM notation). Each resulting 4-byte number is generated in either natural (I64TI4) or byte reversed (I64TWI) order.

Format

```
CALL I64TI4( N , A , B , IXBYTB )
CALL I64TWI( N , A , B , IXBYTB )
```

Parameters

N

Number of Control Data 64-bit integers to convert. Must be of type integer.

A

Input array of type integer, holding N contiguously stored Control Data 64-bit integers. The contents of A are not altered by this routine, provided A and B are not the same array.

B

Output array in which the converted numbers are stored contiguously. Can be of any data type. Each 4-byte result is stored in natural order (0123) when I64TI4 is called, and in byte reversed order (3210) when I64TWI is called. Conversion in place is allowed; that is, B may represent the same address as A. See Remarks below for details about what happens when the value of a Control Data number in A lies outside the range representable in the IEEE Word Integer format.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B where the first converted number should be placed. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Remarks

An IEEE Word Integer is a 32-bit 2's complement integer. The range of numbers representable in this format is [-2147483648,2147483647], inclusive. However, some hardware reserves the most negative number (hexadecimal Z"80000000") for some special purpose, not recognizing it as an integer.

This routine converts integers by simply isolating the lower 32 bits of the Control Data 64-bit integers furnished in array A. Thus, integers outside the range specified above will be converted, but their value will change. For instance, the two integers 2882338816 and -1412628480 (hexadecimal Z"00000000ABCD0000" and Z"FFFFFFFFABCD0000", respectively) are both converted to the Word Integer Z"ABCD0000", with a decimal value of -1412628480.

Subroutine LVCOMP

Purpose

To perform the vector comparison C = A .OP. B, where C is a logical vector, A and B are either scalars or vectors, and .OP. represents one of the four relational operators .EQ., .GE., .LT. and .NE. The type of A and B can be either integer or real. The last two letters of the subroutine names indicate which relational operator is used.

Format

```
CALL LIVEQ ( N , A , LA , B , LB , C , LC )
CALL LIVGE ( N , A , LA , B , LB , C , LC )
CALL LIVLT ( N , A , LA , B , LB , C , LC )
CALL LIVNE ( N , A , LA , B , LB , C , LC )

CALL LRVEQ ( N , A , LA , B , LB , C , LC )
CALL LRVGE ( N , A , LA , B , LB , C , LC )
CALL LRVLT ( N , A , LA , B , LB , C , LC )
CALL LRVNE ( N , A , LA , B , LB , C , LC )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the left operand in the vector comparison. Must be of type integer if LIVEQ, LIVGE, LIVLT, or LIVNE is used, and of type real if LRVEQ, LRVGE, LRVLT, or LRVNE is used.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable.

B

Array (element) indicating the base address of the right operand in the vector comparison. Must be of type integer if LIVEQ, LIVGE, LIVLT, or LIVNE is used, and of type real if LRVEQ, LRVGE, LRVLT, or LRVNE is used.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable.

C

Array (element) indicating the base address of the target vector into which the logical results of the vector comparison are written. Must be of type logical. If A(J*LA).OP.B(J*LB) is true, then C(J*LC) is set to the value .TRUE., which has the internal representation Z"8000000000000000" (hexadecimal). If the condition is false, the corresponding value of C will be set to .FALSE., which is represented by binary 0.

LC

Stride for the C vector. Must be of type integer and nonzero.

Externals — None

Dynamic Space — 512 or 1024 words depending on the values of LA and LB.

Method — Use of the vector comparison instructions CMPEQV, CMPGEV, CMPLTV, and CMPNEV. In LRVGE and LRVLT, a vector floating-point subtraction must precede the actual vector comparison, since only integer vector compare instructions exist. A vector subtraction must also be used whenever LB=0 in LRVGE, LRVLT, LIVGE and LIVLT, since the vector compare instructions can handle scalars only as left, and not as right, operands. This is not a problem in the other routines since the operators .EQ. and .NE. are commutative.

Example

```
PROGRAM MAIN
DIMENSION A(0:49)
LOGICAL L(0:39)
INTEGER STRIDEA, STRIDEB, STRIDEL
LENGTH=40
B=0.0
STRIDEA=1
STRIDEB=0
STRIDEL=1
.
.
CALL LRVLT(LENGTH, A, STRIDEA, B, STRIDEB, L, STRIDEL)
END
```

The call to LRVLT causes the first 40 elements in array A to be compared with the scalar value 0.0. Negative values in A will cause the corresponding element in the logical array L to have the value .TRUE.; positive values or zeroes in A will cause the corresponding element in L to have the value .FALSE.

Subroutime MOVB

Purpose

To move consecutive 8-bit bytes from one location to another.

Format

CALL MOVB(N , A , IXBYTA , B , IXBYTB)

Parameters

N

Number of 8-bit bytes to move. Must be of type integer.

A

Input array holding the N contiguously stored 8-bit bytes. Can be of any data type.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array A of the first byte to be moved. The beginning of A is represented by IXBYTA=0.

B

Output array into which the N consecutive 8-bit bytes are moved. Can be of any data type. If B is the same array as A, care must be taken so that the address defined by B and IXBYTB is smaller than that determined by A and IXBYTA.

IXBYTB

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array B into which the first byte is moved. The beginning of B is represented by IXBYTB=0.

Externals

None

Dynamic Space

None

Method

The hardware instruction MOVB is used, moving 256 bytes per pass in a scalar loop.

Remarks

The MOVB instruction can be regarded as a pseudo vector instruction by virtue of the fact that it handles many items. As a result, on a scalar machine (any CYBER 180 except the CYBER 180-990) it may sometimes be more efficient to call MOVB than SCOPY when the elements of a real or integer array are to be moved.

Subroutine MXADDF

Purpose

To perform the matrix addition C = A + B, where A, B, and C are matrices of type real.

Format

CALL MXADDF (M , N , A , LA , B , LB , C , LC)

Parameters

M,N

Dimensions of each of the three matrices A, B, and C. Both dimension values must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the left operand in the matrix addition. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of the right operand in the matrix addition. Must be of type real.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

C

Array (element) indicating the address of the upper left-hand corner of the result matrix. Must be of type real. C and A may be the same array, provided that LA≥LC. C and B may be the same array, provided that LB ≥LC.

LC

Size of the first dimension (column length) of array C. Must be of type integer and greater than or equal to M.

Externals

None

Dynamic Space

None

Example

```
PARAMETER (LA=2, LB=5, LR=2)
DIMENSION ARRAY1(0:LA-1,0:2),ARRAY2(0:LB-1,0:4),RESULT(0:LR-1,0:5)
IROWS=2
ICOLMNS=3
.
.
CALL MXADDF (IROWS, ICOLMNS, ARRAY1, LA, ARRAY2(2,1), LB, RESULT(0,3), LR)
```

The call to MXADDF causes the array section RESULT(0:1,3:5) to contain the sum of ARRAY1 (all elements) and ARRAY2 (elements (2,1), (3,1), (2,2), (3,2), (2,3), (3,3)).

Subroutine MXCMP

Purpose

To compare two floating-point matrices A and B by computing the maximum relative error of the columns of B with respect to those of A. Both 2-norms and infinity-norms are computed.

Format

CALL MXCMP (M , N , A , LA , B , LB , ERR , TOL)

Parameters

M,N

Dimension of matrices A and B. Both dimension values must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of matrix A, which is considered as the original matrix to which matrix B is to be compared. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of matrix B, which is compared with matrix A. Must be of type real.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

ERR

Output array of length at least 3 words. Must be of type real. On output, ERR contains in its first three locations the maximum relative errors for the N columns as indicated below and described in detail under Method.

ERR(1) = 2-norm

ERR(2) = Infinity-norm

ERR(3) = Maximum value of elements not possible to include in the calculations of the norms.

TOL

Small positive user-supplied tolerance value. Must be of type real. A possible and reasonable choice is 1.E-10. Only elements whose absolute values exceed that of TOL are considered in the calculations of the norms.

Externals

None

Dynamic Space

None

Method

Initially the errors are assigned as ERR(1)=ERR(2)=-1.0, ERR(3)=0.0. The comparison then proceeds with one column at a time. For column K the procedure is as follows:

1. Define the subset jt of the values (0,1,...,M-1) in such a way that MIN(ABS(A(J,K)) , ABS(B(J,K)))>TOL whenever J belongs to jt. Further let jf be defined as the complimentary subset; that is, each value in the set (0,1,...,M-1) is either in jt or in jf.

2. Compute the relative 2-norm of the columns of A and B as

$$E1 = \mathrm{SQRT}\left(\sum_{J\epsilon jt} (A(J,K)-B(J,K))^{**}2 \right) / \left(\sum_{J\epsilon jt} A(J,K)^{**}2 \right)$$

3. Compute the infinity-norm of the relative error vector as

$$E2 = \max_{J\epsilon jf} \left(\mathrm{ABS}(A(J,K)-B(J,K)) / \mathrm{ABS}(A(J,K)) \right)$$

4. Determine the maximum absolute value of the elements not included in the calculations of the norms as

$$E3 = \max\left(\mathrm{TOL} , \max_{J\epsilon jf} \left(\mathrm{ABS}(A(J,K)) , \mathrm{ABS}(B(J,K)) \right) \right)$$

5. Update the values in ERR only if larger errors were found:

```
ERR(1) = MAX( ERR(1) , E1 )
ERR(2) = MAX( ERR(2) , E2 )
ERR(3) = MAX( ERR(3) , E3 )
```

An output value of -1.0 for ERR(1) and ERR(2) thus indicates that there is no subscript (J,K) for which the condition

```
MIN( ABS(A(J,K)) , ABS(B(J,K)) )>TOL
```

holds true. If, on the other hand, the same condition is satisfied for all subscripts (J,K), the final value of ERR(3) is zero.

Function MXENRM

Purpose

To compute the Euclidean norm of a real square matrix A. This is defined as the square root of the sum of the squares of the elements of the matrix.

Format

XENRM = MXENRM (N , A , LA)

Parameters

XENRM

Function result of type real, containing the square root of the sum of the squared elements of A.

N

The order of the input matrix A. Must be of type integer.

A

Array (element) indicating the base address of the input matrix. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer, and be at least as large as N.

Externals

SDOT, SNRM2, SQRT

Dynamic Space

None

Method

This is not a matrix norm in the usual sense of being subordinate to some vector norm. In particular, it is not subordinate to the Euclidean or L-2 vector norm. When LA=N the resulting norm is obtained by means of a single call to the LIB99 function SNRM2. Otherwise the results of N calls to the LIB99 function SDOT are summed, and the intrinsic function SQRT produces the result.

The following computation is performed:

$$XENRM = SQRT \left(\sum_{i,j=0}^{n-1} A(I,J)^{**}2 \right)$$

The FORTRAN equivalent of MXENRM is given below.

```
      REAL FUNCTION MXENRM( N , A , LA )
      REAL A(0:LA-1,0:N-1)
      IF ( LA .EQ. N ) THEN
         MXENRM = SNRM2( N*N , A , 1 )
      ELSE
         MXENRM = 0.0
         DO 10 J = 0 , N-1
         MXENRM = MXENRM + SDOT( N , A(0,J) , 1 , A(0,J) , 1 )
10       CONTINUE
         MXENRM = SQRT( MXENRM )
      ENDIF
      RETURN
      END
```

Function MXEQ

Purpose

To determine whether two matrices are identical, using a bit-by-bit method of comparison.

Format

```
ANSWER = MXEQ ( M , N , A , LA , B , LB )
```

Parameters

ANSWER

The function result is of type logical. The value .TRUE. results if A(J,K)=B(J,K) for all (J,K) such that J=0,1,...,M-1 and K=0,1,...,N-1. The value .FALSE. results if for at least one pair of subscripts A(J,K)≠B(J,K).

M,N

Dimension of matrices to be compared. Both dimension values must be of type integer.

A

Array (element) indicating upper left-hand corner of the first matrix. Can be of any type except character.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating upper left-hand corner of the second matrix. Can be of any type except character.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

Externals

None

Dynamic Space

None

Subroutine MXGEFS

Purpose

To solve a system of linear equations; that is, the matrix equation AX=B, where A is a full matrix using a form of Gaussian elimination.

Format

CALL MXGEFS (N , NRHS , A , LA , B , LB , IPIV , DET , EPS , IOPT)

Parameters

N

Number of equations to be solved, or order of the matrix A. Must be of type integer.

NRHS

Number of right-hand sides for which to solve. NRHS must be a positive integer when substitution is requested (see IOPT below) but is ignored otherwise.

A

Input/output array holding the square matrix A of order N as a submatrix. Must be of type real.

INPUT: If decomposition is requested (IOPT is odd), the array should contain the coefficients that define the left-hand side coefficient matrix A in the matrix equation AX=B.

If decomposition is not requested (IOPT is even), the content of A should be exactly what was produced by a previous call to this routine, in which decomposition was indeed requested.

OUTPUT: If decomposition is requested (IOPT is odd), the array contains the result of that decomposition; that is, information sufficient to define the L and the U in the LU decomposition of A. This form of A can be saved by the caller and later used for repeated calls to this routine with different right-hand sides. Thus, a given matrix A need not be decomposed more than once.

If decomposition is not requested (IOPT is even), the content of array A is not altered by this call.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to N.

B

Input/output array of type real. Meaningful only if substitution is requested; that is, if AND(IOPT,2)≠0. The contents of B are defined as follows:

INPUT: The NRHS right-hand sides for which to be solved, one per column.

OUTPUT: The corresponding solutions, similarly stored.

Parameters (Continued)

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to N.

IPIV

Input/output scratch array of length at least 2N words. Can be of any type except character. If decomposition is requested (IOPT is odd) the input content of IPIV is of no consequence. However, in this case this routine produces information necessary for a subsequent call where decomposition is not requested, and stores that in IPIV. Thus, if a later call with IOPT even is planned, IPIV must be saved and supplied as an input array at that time.

DET

Array of length at least 2 words. Must be of type real. Meaningful only if decomposition is requested; that is, if IOPT is odd, in which case the determinant of the coefficient matrix A is computed and stored in array DET.

DET(0) can always be interpreted as a singularity flag; DET(0)=0.0 indicates that the matrix A, possibly due to rounding errors, was found to be singular. A nonzero value for DET(0) indicates that the decomposition proceeded smoothly. The user should always examine the value of DET(0) after each call since a singular matrix results in all arrays containing incorrect data upon return.

Since the determinant computation for very large matrices easily generates floating-point overflow, more than one word may sometimes be needed to report the result. Although the routine is designed so that overflow will not occur, you must use care in dealing with the decomposed value of the determinant, as stored in array DET.

If the final value of the determinant can be represented as one 64-bit floating-point value, this is done, and the value is stored in DET(0). This case is identified by DET(1)=0.0, a condition that can be checked by the user. If, on the other hand, the absolute value of the determinant is too large, both words of array DET must be used for its representation: DET(0) contains a floating-point number whose absolute value lies in the range (1.0,2.0), while the remainder of the exponent is stored as a 64-bit floating-point number in DET(1). Thus, in this case DETERMINANT=(2**IFIX(DET(1)))*DET(0). Note that actually performing the above calculation generates an overflow as soon as DET(1) is nonzero.

EPS

Small positive user-supplied number, used for singularity tests during the decomposition phase. Must be of type real. The value of EPS sould be small compared to the range of values of the entries in the coefficient matrix A, but large enough to provide a meaningful definition of zero. If the absolute values of the nonzero matrix entries are all in the range ALFA to 2*ALFA, then a reasonable value for EPS is probably 1.E-10*ALFA. See the description of Gaussian elimination under Method.

Parameters (Continued)

IOPT

IOPT=I(0)+I(1)+I(2)+I(3), where I(J) can either take the value 0 or 2**J, depending on whether a specific option is requested (2**J) or not (0). The various options and the values for a request are defined below.

I(0)= 1 requests matrix decomposition.
I(1)= 2 requests forward and backward substitution.
I(2)= 4 requests partial pivoting.
I(3)= 8 requests scaling **CURRENTLY NOT IMPLEMENTED**.

The different processes are defined under Method. As an example, a complete solution of AX=B, implementing pivoting but not scaling, would be requested by IOPT=7.

Externals

VGATHR

Dynamic Space

1102 words.

Method

Gaussian elimination, as described in most standard textbooks on the topic of solving systems of linear equations. One such reference is: J. H. Wilkinson: The Algebraic Eigenvalue Problem, Oxford University Press, London (1965), pages 200-217. For a given call one or several of the steps described below would be implemented.

1. Scaling **CURRENTLY NOT IMPLEMENTED**

   Although it is difficult to make a general statement for all types of matrices, it is widely believed that a good way of improving the accuracy of the solutions is to initially scale the left-hand side matrix A in such a way that the absolute values of all entries are forced into some arbitrary interval close to 1.0. The interval here is chosen as ABS(A(J,K))<0.5.

   The matrix A is first scanned in order to find the largest exponent in each row. A scaling vector S with N elements is then prepared as follows:

   The coefficient of S(J) will be the same as that of 1.0; the exponent will be such that multiplication of each element in row J with S(J) will yield scaled elements with absolute values all strictly less than 0.5. By assembling the scaling vector in such a way we ascertain that the subsequent process of scaling will not introduce extra rounding errors - all elements of S are powers of two, so that multiplication merely amounts to an adjustment of exponents.

   While it is necessary to access each element of the matrix once in order to determine the elements of the scaling vector, that is not true as far as the scaling itself is concerned. Rather, a given column of the matrix is not scaled until just before its subdiagonal elements are eliminated during the decomposition phase. The right-hand sides are scaled the first time they are accessed during the forward substitution phase. Thus, for large values of N, matrix A is paged through exactly one extra time if scaling is requested, while no extra paging is required for the right-hand sides. No rounding errors are introduced.

Method (Continued)

2. Decomposition

For each column K, K traversing the values 0 through N-2, the following steps are required in order to decompose the matrix A:

2.1 Pivoting

2.1.1 Search through the elements (A(J,K), J=K,...,N-1) and determine the subscript I such that A(I,K) is the element with the largest absolute value.

2.1.2 Perform the pivoting; that is, exchange the elements of row I with those of row K. This swap has to affect all the columns, even those to the left of column K.

2.2 Test for singularity. This amounts to executing the statement IF (ABS(A(K,K)) .LE. EPS) RETURN, where the value of DET(0) has been preset to zero. The user versed in the topics of numerical stability and ill-conditioned matrices can use this information to determine the optimal value for the parameter EPS.

2.3 Create column multipliers. This amounts to performing the vector multiplication A(K+1:N-1,K)=(1.0/A(K,K))*A(K+1:N-1,K).

2.4 For all J such that J=K+1,...,N-1, adjust column J as follows: A(K+1:N-1,J)=A(K+1:N-1,J)-A(K,J)*A(K+1:N-1,K). This is the columnwise analogy of the rowwise oriented process of subtracting an L-dependent fraction of row K from each of the rows L such that L=K+1,...,N-1 which is commonly described in textbooks.

3. Forward substitution

The code for this process is embedded in the code for the decomposition; the paging requirements are therefore minimized.

3.1 Scale and/or pivot as required.

3.2 Adjust the right-hand sides. This is analogous to step 2.4 above. For each K such that K=0,...,N-2, and each J such that J=0,...,NRHS-1, perform the following operation: B(K+1:N-1,J)=B(K+1:N-1,J)-B(K,J)*A(K+1:N-1,K).

4. Backward substitution

Consider only the upper triangular part of matrix A, now containing the elements of the upper triangular matrix U in the LU decomposition of A, and solve the triangular system of linear equations.

The method here is to work backwards and determine X(K) in order of decreasing subscripts K: K=N-1,...,0. For a given step K the operations to perform for each right-hand side J, J=0,...,NRHS-1, are:

4.1 B(K,J)=B(K,J)/A(K,K).
This determines the Kth unknown of the Jth right-hand side.

4.2 B(0:K-1,J)=B(0:K-1,J)-B(K,J)*A(0:K-1,K).
B(K,J) is thus substituted back into A, thereby eliminating that unknown from the system of equations. Since we need to keep A unaltered, the effect of this substitution is directly taken into account by subtracting the result of this vector multiplication from the right-hand side currently under consideration.

Remarks

As can be inferred from the above description, all steps except one manipulate elements of data that are stored contiguously in memory, making the algorithm very well suited for vectorization. The exception is the pivoting process, in which rows are interchanged. Fortunately that is a process which can be efficiently handled by invoking the hardware periodic GATHER and SCATTER instructions. But even if that had not been the case, we would not have suffered since the pivoting is a low order process; that is, compared to the total operation count for the algorithm, the pivoting contributes very little.

Subroutime MXINVU

Purpose

To compute the upper triangular inverse R of a nonsingular upper triangular matrix U.

Format

CALL MXINVU (N , U , KU , R , KR , D)

Parameters

N

The order of the input matrix U. Must be of type integer.

U

Input array containing the upper triangular part of the upper triangular matrix U. Must be of type real. The contents of U are not altered by this routine unless U and R are the same array, which is permissible.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}\ u_{01}\ u_{11}\ u_{02}\ u_{12}\ u_{22}\ u_{03}\ \cdots$$

A value greater than or equal to N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the values of the subdiagonal elements are irrelevant; only the upper triangular part is used.

R

Output array to receive the nontrivial (upper triangular) part of the upper triangular inverse of matrix U. Must be of type real. Regardless of the value of KR, only the upper triangular part of R is written, leaving the lower part untouched. If U and R are the same array, which is permissible, KU=KR must hold.

KR

Storage mode indicator for R. Must be of type integer. A value of 1 indicates linear storage; a value greater than or equal to N indicates two-dimensional storage in an array where the size of the first dimension (column length) is KR. See also the description of KU above.

D

Input array of length at least N words, which must contain the reciprocals of the diagonal elements of U in its first N locations. Must be of type real. On output, the contents of D will have been destroyed.

Externals

LOCF, RSMULV, RVAXPY

Dynamic Space

None

Method

Let the upper triangular matrix U be stored in a two-dimensional array, also denoted by U. Assume that U is nonsingular, and denote its inverse (also upper triangular) by R. Letting P represent the product RU, we then have

$$P(i,j) = \sum_{k=i}^{j} (R(i,k) * U(k,j)) = \begin{cases} 1 \text{ if } i=j \\ 0 \text{ if } i \neq j \end{cases}$$

To solve for R(i,j) we isolate the k=j term

$$R(j,j) = 1.0 / U(j,j)$$

$$R(i,j) = (-1.0 / U(j,j)) * \sum_{k=i}^{j-1} (R(i,k) * U(k,j)) \qquad [i<j]$$

Introducing the auxiliary array H, we start the computation of column j with the vector multiplication

$$H(0:j-1) = R(0:j-1,j-1) * U(j-1,j)$$

accomplished by the call

```
CALL RSMULV ( J , U(J-1,J) , R(0,J-1) , H )
```

Next we must compute j-1 linked triads, k taking the values j-2,j-3,...,0:

$$H(0:k) = H(0:k) + R(0:k,k) * U(k,j)$$

accomplished by the call

```
CALL RVAXPY ( K+1 , U(K,J) , R(0,K) , H , H )
```

Finally we must compute R(j,j) and perform one more vector multiplication, this time with column j itself as a target:

$$R(j,j) = 1.0 / U(j,j)$$
$$R(0:j-1,j) = - R(j,j) * H(0:j-1)$$

The latter statement is executed by means of the call

```
CALL RSMULV ( J , -R(J,J) , H , R(0,J) )
```

The division can be omitted since the reciprocals of the diagonal elements are furnished on input in array D.

A careful analysis of the formulas and code above reveals that in order to make it possible to let U and R mean the same physical arrays, we must proceed with the computations from left to right; that is, column j must be computed before column j+1. This also permits us to let H and D be the same array.

The routine MXINVU, in its basic form, does not differ much from what is outlined above. Thus, in particular, it is almost completely vectorized, with an average vector length of N/3.

Subroutine MXMAB

Purpose

To perform the matrix multiplication C=AB, where C, A, and B are rectangular matrices of matching dimensions.

Format

CALL MXMAB (K , L , M , A , LA , B , LB , C , LC)

Parameters

K

First dimension of matrices A and C. Must be of type integer.

L

Second dimension of matrix A, and the first dimension of matrix B. Must be of type integer.

M

Second dimension of matrices B and C. Must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of matrix A, the left multiplicand. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to K.

B

Array (element) indicating the address of the upper left-hand corner of matrix B, the right multiplicand. Must be of type real.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to L.

C

Array (element) indicating the address of the upper left-hand corner of matrix C, the product (result) matrix. Must be of type real. Matrix C must not overlap with either one of the two input matrices A and B.

LC

Size of the first dimension (column length) of array C. Must be of type integer and greater than or equal to K.

Externals

None

Dynamic Space

512 words.

Method

A completely vectorized outer product method is used.

Example

```
DIMENSION A(0:9,0:9), B(0:7,0:7), RESULT(0:2,0:4)
.
.
.
CALL MXMAB (3, 5, 5, A, 10, B(2,3), 8, RESULT, 3)
```

The execution of the call to MXMAB causes the [3,5] matrix in array A to be multiplied from the right by the [5,5] matrix in array B, and the resulting [3,5] matrix to be stored in array RESULT. The three matrices occupy the location described by A(0:2,0:4), B(2:6,3:7), and RESULT(0:2,0:4), respectively.

Subroutine MXMOVF

Purpose

To move a matrix from one location to another.

Format

```
CALL MXMOVF ( M , N , A , LA , B , LB )
```

Parameters

M,N

Dimensions of the matrix to be moved. Both dimension values must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the source matrix. Can be of any type except character.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of the target matrix. Must be of the same type as A. B and A may be the same array, provided that LA≥LB.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

Externals

None

Dynamic Space

None

Method

Repeated uses of the vector instruction SHFV.

Example

```
DIMENSION J(0:9,0:14), K(0:9,0:19)
.
.
.
CALL MXMOVF(5, 5, J(1,3), 10, K(1,1), 10)
```

The result of the execution of the MXMOVF call statement is that the 25 elements of the [5,5] matrix stored in the array section J(1:5,3:7) are copied into the array section K(1:5,1:5).

Subroutine MXMOVU

Purpose

To move the upper triangular matrix U from one location to another, optionally changing the storage mode from columnwise linear (standard symmetric) to two-dimensional, or vice versa.

Format

CALL MXMOVU (N , U , KU , R , KR)

Parameters

N

The order of the source matrix U. Must be of type integer.

U

Input array containing the upper triangular part of the upper triangular matrix U. Can be of type integer or real.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}-u_{01}-u_{11}-u_{02}-u_{12}-u_{22}-u_{03}-\cdots$$

A value greater than or equal to N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the subdiagonal elements are ignored; only the upper triangular part, or N(N+1)/2 elements, are moved.

R

Output array to which the upper triangular part of U is moved. Must be of the same type as U. Regardless of the value of KR, only the upper triangular part of U is moved, leaving the lower part of R untouched when KR≥N.

KR

Storage mode indicator for R. Must be of type integer. A value of 1 indicates linear storage; a value greater than or equal to N indicates two-dimensional storage in an array where the size of the first dimension (column length) is KR. See also the description of KU above.

Externals

SCOPY

Dynamic Space

None

Method

Repeated calls to SCOPY.

Subroutine MXMUTU

Purpose

To perform the matrix multiplication S=LU, where L is the transpose of the supplied upper triangular matrix U, and S is the symmetric product matrix.

Format

CALL MXMUTU (N , U , KU , S , KS)

Parameters

N

The order of the input matrix U. Must be of type integer.

U

Input array containing the upper triangular part of the upper triangular matrix U. Must be of type real. The contents of U are not altered by this routine unless U and S are the same array, which is permissible.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}\ u_{01}\ u_{11}\ u_{02}\ u_{12}\ u_{22}\ u_{03}\ \cdots$$

A value greater than or equal to N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the values of the subdiagonal elements are irrelevant; only the upper triangular part is used.

S

Output array to receive the nontrivial (upper triangular) part of the symmetric product matrix S. Must be of type real. Regardless of the value of KS, only the upper triangular part of S is written, leaving the lower part untouched. If U and S are the same array, which is permissible, KU=KS must hold.

KS

Storage mode indicator for S. Must be of type integer. A value of 1 indicates linear storage; a value greater than or equal to N indicates two-dimensional storage in an array where the size of the first dimension (column length) is KS. See also the description of KU above.

Externals

LOCF, SDOT

Dynamic Space

None

Method

Let the upper triangular matrix U be stored in a two-dimensional array, also denoted by U. Letting L denote the transpose of U and forming the product S=LU, we obtain

$$S(i,j) = \sum_{k=0}^{i} (L(i,k) * U(k,j)) = \sum_{k=0}^{i} (U(k,i) * U(k,j)) \quad [i \le j]$$

Clearly S is symmetric, so we only need to consider the upper triangular part of the product.

Method (Continued)

If we require that S and U should be allowed to occupy the same physical memory space, the order of computation becomes uniquely determined: start with column n-1 and compute each element there as a vectorized dot product, work upwards; that is, don't start with element (j,n-1) until element (j+1,n-1) of S has been computed. Then move to column n-2, etc.

To implement the formula above we would thus create a nested loop where the only nontrivial operation would be

$$S(i,j) = \sum_{k=0}^{i} (U(k,i) * U(k,j))$$

accomplished by the call

```
S(I,J) = SDOT( I+1 , U(0,I) , 1 , U(0,J) , 1 )
```

The inner loop would be controlled by the loop index i, decreasing from j to 0. The outer loop would be controlled by j, which would decrease from n-1 to 0.

MXMUTU, in its basic form, does not differ much from what is outlined above. Thus, in particular, almost all work is performed in vector mode, using an average vector length of N/3.

As is evident, the computation of column j requires the presence in memory of all the columns k of U such that k=j,j-1,...,0. That implies that we will have a thrashing situation if all of array U does not fit in available core.

Subroutine MXMUUT

Purpose

To perform the matrix multiplication S=UL, where L is the transpose of the supplied upper triangular matrix U, and S is the symmetric product matrix.

Format

CALL MXMUUT (N , U , KU , S , KS)

Parameters

N

The order of the input matrix U. Must be of type integer.

U

Input array holding the upper triangular part of the upper triangular matrix U. Must be of type real. The contents of U are not altered by this routine unless U and S are the same array, which is permissible.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of U are stored linearly, thus omitting the subdiagonal elements, which are all zeroes:

$$u_{00}\text{-}u_{01}\text{-}u_{11}\text{-}u_{02}\text{-}u_{12}\text{-}u_{22}\text{-}u_{03}\text{-}\cdots$$

A value greater than or equal to N indicates full storage mode; that is, KU is the size of the first dimension (column length) of the two-dimensional array U. In this case the values of the subdiagonal elements ae irrelevant; only the upper triangular part is used.

S

Output array to receive the nontrivial (upper triangular) part of the symmetric product matrix S. Must be of type real. Regardless of the value of KS, only the upper triangular part of S is written, leaving the lower part untouched. If U and S are the same array, which is permissible, KU=KS must hold.

KS

Storage mode indicator for S. Must be of type integer. A value of 1 indicates linear storage; a value greater than or equal to N indicates two-dimensional storage in an array where the size of the first dimension (column length) is KS. See also the description of KU above.

Externals

LOCF, RSMULV, RVAXPY

Dynamic Space

None

Method

Let the upper triangular matrix U be stored in a two-dimensional array, also denoted by U. Letting L denote the transpose of U and forming the product S=UL, we immediately obtain

$$S(i,j) = \sum_{k=j}^{n-1} (U(i,k) * L(k,j)) = \sum_{k=j}^{n-1} (U(i,k) * U(j,k)) \qquad [i \leq j]$$

Clearly S is symmetric, so we only need to consider the upper triangular part of the product.

If we require that S and U should be allowed to occupy the same physical memory space, the order of compulation becomes uniquely determined: proceed from left to right, computing column 0 first, then column 1, and so forth.

Method (Continued)

To compute column j we must first multiply the vector (U(i,j),i=0,...,j) by the scalar element U(j,j):

```
S(0:j,j) = U(0:j,j) * U(j,j)
```

accomplished by the call

```
CALL RSMULV ( J+1 , U(J,J) , U(0,J) , S(0,J) )
```

To obtain the final value for column j we must compute n-j-1 linked triad operations, letting k run from j+1 to n-1:

```
S(0:j,j) = S(0:j,j) + U(0:j,k) * U(j,k)
```

accomplished by the call

```
CALL RVAXPY( J+1, U(J,K) , U(0,K) , S(0,J) , S(0,J) )
```

MXMUUT, in its basic form, does not differ much from what is outlined above. Thus, in particular, almost all work is performed in vector mode, using an average vector length of N/3.

As is evident, the computation of column j requires the presence in memory of all the columns k of U such that k=j,j+1,...,n-1. That implies that we will have a thrashing situation if all of array U does not fit in available core.

Subroutine MXSCAF

Purpose

To scale the elements of a real matrix; that is, to multiply each element with a constant real value.

Format

```
CALL MXSCAF ( M , N , X , A , LA , B , LB )
```

Parameters

M,N

Dimensions of the matrix to be scaled. Both dimension values must be of type integer.

X

Real scalar constant or variable, or array element, with which each element in the matrix A will be multiplied.

A

Array (element) indicating the address of the upper left-hand corner of the source matrix. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of the scaled matrix. Must be of type real. B may be the same array as A, provided that LA≥LB.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

Externals

None

Dynamic Space

None

Example

```
DIMENSION ARRAY1(0:49,0:99), ARRAY2(0:49,0:99)
.
.
.
CALL MXSCAF(50, 100, 2.0, ARRAY1, 50, ARRAY2, 50)
```

After the execution of the call to MXSCAF, all locations of ARRAY2 contain twice the value of the corresponding locations of ARRAY1.

Subroutine MXSUBF

Purpose

To perform the matrix subtraction C = A - B, where A, B, and C are matrices of type real.

Format

CALL MXSUBF (M , N , A , LA , B , LB , C , LC)

Parameters

M,N

Dimensions of each of the three matrices A, B and C. Both dimension values must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the left operand in the matrix subtraction. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of the right operand in the matrix subtraction. Must be of type real.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to M.

C

Array (element) indicating the address of the upper left-hand corner of the result matrix. Must be of type real. C and A can be the same array, provided that LA≥LC. C and B can be the same array, provided that LB≥LC.

LC

Size of the first dimension (column length) of array C. Must be of type integer and greater than or equal to M.

Externals

None

Dynamic Space

None

Example

```
DIMENSION X(0:9,0:9), Y(0:9,0:9), Z(0:9,0:9)
.
.
.
CALL MXSUBF(10, 10, X, 10, Y, 10, Z, 10)
```

The call to MXSUBF will, for all values of I and J ranging from 0 through 9, cause the value Y(I,J)-X(I,J) to be written into location Z(I,J).

Subroutine MXSYUL

Purpose

To perform the Cholesky decomposition A = UL of the positive definite symmetric matrix A. U denotes a nonsingular upper triangular matrix, and L represents its lower triangular transpose. Note that the order of the factors U and L is reversed from that in the conventional LU decomposition. This is done in order to gain maximum performance on the CYBER 990.

Format

CALL MXSYUL (N , A , KA , U , KU , D , NERR)

Parameters

N

The order of the input matrix A. Must be of type integer.

A

Input array containing the upper triangular part of the positive definite symmetric matrix A. Must be of type real. The contents of A are not altered by this routine unless A and U are the same array, which is permissible.

KA

Storage mode indicator for A. Must be of type integer. A value of 1 indicates that the columns of the upper triangular part of A are stored linearly, thus omitting the subdiagonal elements:

$$a_{00}-a_{01}-a_{11}-a_{02}-a_{12}-a_{22}-a_{03}-\cdots$$

A value greater than or equal to N indicates full storage mode; that is, KA is the size of the first dimension (column length) of the two-dimensional array A. In this case the values of the subdiagonal elements are irrelevant; only the upper triangular part is used.

U

Output array to receive the nontrivial (upper triangular) part of the upper triangular matrix U. Must be of type real. Regardless of the value of KU, only the upper triangular part of U is written, leaving the lower part untouched. If A and U are the same array, which is permissible, KA=KU must hold.

KU

Storage mode indicator for U. Must be of type integer. A value of 1 indicates linear storage; a value greater than or equal to N indicates two-dimensional storage in an array where the size of the first dimension (column length) is KU. See also the description of KA above.

D

Output array of length at least N words, which receives the reciprocals of the diagonal elements of U in its first N locations. Must be of type real.

NERR

Error flag set by this routine. Must be of type integer. A value of 0 indicates that no errors were incurred. A value of 1 indicates that the matrix A (possibly due to rounding errors) proved not to be positive definite, in which case U and D contain incorrect data.

Externals

LOCF, RSMULV, RVAXPY

Dynamic Space None

Method

Let the positive definite symmetric matrix to be decomposed be denoted by A, and assume that it is stored in a two-dimensional array, also denoted by A. By slightly modifying a well-known theorem by Cholesky, it can be shown that A can be decomposed into the product of an upper triangular matrix U and its lower triangular transpose L; that is, A=UL. From this we directly obtain

$$A(i,j) = \sum_{k=0}^{n-1} (U(i,k) * L(k,j)) = \sum_{k=j}^{n-1} (U(i,k) * U(j,k)) \qquad [i \leq j]$$

To solve for U(i,j) we isolate the k=j term

$$U(j,j) = SQRT(A(j,j) - \sum_{k=j+1}^{n-1} (U(j,k)**2))$$

$$U(i,j) = (1/U(j,j)) * (A(i,j) - \sum_{k=j+1}^{n-1} (U(i,k)*U(j,k)) \qquad [i<j]$$

The first step in computing column j of U is to compute the following linked triad:

```
U(0:j,j) = A(0:j,j) - U(0:j,j+1) * U(j,j+1)
```

accomplished by the call

```
CALL RVAXPY ( J+1, -U(J,J+1), U(0,J+1), A(0,J), U(0,J) )
```

Having performed this operation, we no longer need A for the purpose of computing the elements in column j of U. We proceed with computing an additional n-j-2 linked triads of the following type, letting k take the values j+2,j+3,...,n-1:

```
U(0:j,j) = U(0:j,j) - U(0:j,k) * U(j,k)
```

accomplished by the call

```
CALL RVAXPY( J+1 , -U(J,K) , U(0,K) , U(0,J) , U(0,J) )
```

To complete the computation of column j we need to perform the following steps, where array D is used to hold the reciprocals of the diagonal elements:

```
U(j,j)     = SQRT( U(j,j) )
D(j)       = 1.0 / U(j,j)
U(0:j-1,j) = D(j)*U(0:j-1,j)
```

The last of these three operations can be accomplished by the call

```
CALL RSMULV (J, D(J), U(0,J), U(0,J))
```

A careful analysis of the formulas and code above reveals that in order to make it possible to let A and U mean the same physical arrays, we must proceed with the computations from right to left; that is, column j must be computed before column j-1.

The routine MXSYUL, in its basic form, does not differ much from what is outlined above. Thus, in particular, it is almost completely vectorized, with an average vector length of N/3.

Subroutine MXTRAF

Purpose

To transpose a rectangular matrix, simultaneously moving it from one location to another.

Format

CALL MXTRAF (M , N , A , LA , B , LB)

Parameters

M,N

Dimensions of the matrix to be transposed. Both dimension values must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the source matrix. Can be of any type except character. The used portions of arrays A and B must not overlap.

LA

Size of the first dimension (column length) of array A. Must be of type integer and greater than or equal to M.

B

Array (element) indicating the address of the upper left-hand corner of the target for the transposed matrix. Must be of the same type as A. The used portions of arrays A and B must not overlap.

LB

Size of the first dimension (column length) of array B. Must be of type integer and greater than or equal to N.

Externals

None

Dynamic Space

None

Method

When $M \leq N$ the periodic gather instruction GTHV is used, while in other cases the periodic scatter instruction SCTV is employed. The work performed by MXTRAF is equivalent to that performed when the following FORTRAN DO loop is executed:

```
      DO 10 K=0,N-1
      DO 10 J=0,M-1
      B(K,J)=A(J,K)
   10 CONTINUE
```

Subroutine MXTRAU

Purpose

To transpose the upper triangular matrix U, thus creating the lower triangular part of a lower triangular matrix L. If U and L are the same array, this amounts to expanding a symmetric matrix where only the upper triangular part is stored into a form where all N*N elements are represented.

Format

CALL MXTRAU (N , U , KU , L , KL)

Parameters

N

The order of the source matrix U. Must be of type integer.

U

Input array containing the upper triangular part of the upper triangular matrix U. Can be of any type except character.

KU

Size of the first dimension (column length) of array U. Must be of type integer and greater than or equal to N.

L

Output array, into which the lower triangular part of the transpose of the upper triangular matrix U is written. Must be of the same type as U. If U and L are the same array, which is permissible, KU=KL must hold. Note that the superdiagonal elements of L are not altered by this routine.

KL

Size of the first dimension (column length) of array L. Must be of type integer and greater than or equal to N. KL must equal KU if U and L are the same array.

Externals

VGATHP

Dynamic Space

None

Method

Currently the work is done by means of repeated calls to VGATHP. The execution of the following FORTRAN DO loop would accomplish the same result as a call to MXTRAU:

```
      DO 10 K=0,N-1
      DO 10 J=0,K
      L(K,J)=U(J,K)
10    CONTINUE
```

Function MX1NRM

Purpose

To compute the 1-norm of a real square matrix A. This is defined as the largest sum of absolute values of the elements in any one column, and is sometimes referred to as the maximum column sum.

Format

X1NRM = MX1NRM (N , A , LA)

Parameters

X1NRM

Function result of type real, containing the 1-norm of A.

N

The order of the input matrix A. Must be of type integer.

A

Array (element) indicating the base address of the input matrix. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer, and be at least as large as N.

Externals

RVASUM

Dynamic Space

None

Method

This is a matrix norm in the usual sense of being subordinate to a vector norm. In particular, it is subordinate to the L-1 (or sum) vector norm. The LIB99 function RVASUM is called N times to determine the maximum column sum.

The following computation is performed:

$$X1NRM = \max_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} \left(ABS(A(I,J)) \right) \right)$$

The FORTRAN equivalent of MX1NRM is given below.

```
      REAL FUNCTION MX1NRM( N , A , LA )
      REAL A(0:LA-1,0:N-1)
      MX1NRM = 0.0
      DO 10 J = 0 , N-1
      X = RVASUM (  N , A(0,J) , 1 )
      MX1NRM = MAX ( MX1NRM , X )
10    CONTINUE
      RETURN
      END
```

Function MX8NRM

Purpose

To compute the infinity-norm of a real square matrix A. This is defined as the largest sum of absolute values of the elements in any one row, and is sometimes referred to as the maximum row sum.

Format

```
X8NRM = MX8NRM ( N , A , LA )
```

Parameters

X8NRM

Function result of type real, containing the infinity-norm of A.

N

The order of the input matrix A. Must be of type integer.

A

Array (element) indicating the base address of the input matrix. Must be of type real.

LA

Size of the first dimension (column length) of array A. Must be of type integer, and be at least as large as N.

Externals

RVASUM

Dynamic Space

None

Method

This is a matrix norm in the usual sense of being subordinate to a vector norm. In particular, it is subordinate to the L-infinity (or max) vector norm. The LIB99 function RVASUM is called N times to determine the maximum row sum.

The following computation is performed:

$$X8NRM = \max_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} \left(ABS(A(I,J)) \right) \right)$$

The FORTRAN equivalent of MX8NRM is given below.

```
      REAL FUNCTION MX8NRM( N , A , LA )
      REAL A(0:LA-1,0:N-1)
      MX8NRM = 0.0
      DO 10 J = 0 , N-1
      X = RVASUM (  N , A(J,0) , LA )
      MX8NRM = MAX ( MX8NRM , X )
10    CONTINUE
      RETURN
      END
```

Subroutine QSORT

Purpose

To arrange (sort) the elements of a real (QSORT) or integer (IQSORT) vector in non-decreasing order with respect to algebraic value: $A(J) \leq A(J+1)$ on return.

Format

```
CALL IQSORT ( N , A, K )
CALL QSORT ( N , A , K )
```

Parameters

N

Number of elements to be ordered. Must be of type integer.

A

Array (element) indicating the base address of the list whose elements are to be ordered. Must be of type real when QSORT is called, and of type integer when IQSORT is called.

K

Output array of length at least N words. Must be of type integer. Array K must not follow immediately after array A; that is, A(N) and K(0) must not represent the same memory location. On return from QSORT, K contains an index list reflecting the rearrangement of the elements of A. As a specific example, assume that A originally contained the three elements a_0, a_1, and a_2, in that order, and that they after sorting by QSORT ended up in the order a_2, a_0, a_1. The first three locations of K would then contain the integers 2, 0, and 1.

If an array B was associated with the original array A in such a way that for each index J, B(J) was associated with A(J), then this situation could be restored using a temporary array C as follows:

```
CALL VGATHR ( N , B , K , C )
CALL SCOPY  ( N , C , 1 , B , 1 )
```

Externals

None

Dynamic Space

None

Method

Vectorized median-of-3 quicksort, as described in all standard textbooks on sorting. This method is unstable; that is, elements that have the same value may change order.

Example

```
DIMENSION ARAY(0:149), INDEX(0:149)
.
.
.
CALL QSORT(150, ARAY, INDEX)
```

After the call to QSORT, the array ARAY contains the sorted values. The index list is contained in array INDEX.

Subroutine R4TF64

Purpose

To convert one or several IBM 32-bit floating-point numbers, also known as REAL*4, to Control Data 64-bit floating-point format.

Format

CALL R4TF64 (N , A , IXBYTA, B)

Parameters

N

Number of IBM 32-bit floating-point numbers to convert. Must be of type integer.

A

Input array holding the N contiguously stored IBM 32-bit numbers (two per 64-bit word) that are to be converted. Can be of any type except character. The contents of A are not altered by this routine.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be of type integer. Specifies the location in array A of the leftmost byte of the first number to be converted. The beginning of A is represented by IXBYTA=0.

B

Output array in which the Control Data 64-bit floating-point numbers are stored. Must be of type real. Conversion in place is not allowed; that is, B and A must be distinct. No special IBM numbers, such as indefinites or infinities, are recognized. Consequently, all numbers generated in array B are standard floating-point numbers.

Externals

None

Dynamic Space

1024 words.

Remarks

The IBM 32-bit floating-point format features a sign bit (0=plus, 1=minus) to the far left, followed by a 7-bit exponent expressed in base 16 and biased by 64 (making it positive). The rightmost 24 bits constitute a normalized positive coefficient. The normalization is done 4 bits at a time, so that the most significant 4 bits of the coefficient may form any hexadecimal digit except 0.

Function SASUM
(from the BLAS package)

Purpose

To compute the sum of the absolute values of the elements of a vector of type real. See BLAS conventions in the introduction.

Format

ASUM=SASUM (N , A , LA)

Parameters

ASUM

Function result of type real, containing the sum of the absolute values of the elements of A.

N

Number of elements to be summed. Must be of type integer.

A

Array (element) indicating the base address of the input vector. Must be of type real.

LA

Stride for the A vector. Must be of type integer. Only the magnitude of LA is significant. Negative values are negated before use.

Externals

None

Dynamic Space

512 words.

Method

The hardware vector instruction SUMFV is used. The actual summing is preceded by a conversion to absolute values, handled by the vector instruction ANDV.

NOTE

This routine is very similar to the LIB99 function RVASUM (see VSUM).

Subroutine SAXPY
(from the BLAS package)

Purpose

To compute the linked triad Y=A*X+Y, where A is a scalar and X and Y are vectors. See BLAS conventions in the introduction.

Format

CALL SAXPY (N , A , X , LX , Y , LY)

Parameters

N

Number of elements in the vectors X and Y. Must be of type integer.

A

Scalar constant or variable or array element, which is used to initially scale the elements of the vector X. Must be of type real.

X

Array (element) indicating the base address of the first input vector. Must be of type real.

LX

Stride for the X vector. Must be of type integer and not equal to zero. See BLAS conventions in the introduction for interpretation of the LX<0 case.

Y

Array (element) indicating the base address of the second input vector which is also the result vector. Must be of type real.

LY

Stride for the Y vector. Must be of type integer and not equal to zero. See BLAS conventions in the introduction for interpretation of the LY<0 case.

Externals

None

Dynamic Space

512 or 1024 words, depending on the values of LX and LY.

Method

Vector multiplication, using the vector instruction MULFV, followed by vector addition, using the vector instruction ADDFV.

NOTE

This routine is very similar to the LIB99 subroutine RVAXPY (see VAXPY).

Subroutine SCOPY
(from the BLAS package)

Purpose

To move the elements of a vector from one location to another. See BLAS conventions in the introduction.

Format

CALL SCOPY (N , A , LA , B , LB)

Parameters

N

Number of elements to move. Must be of type integer.

A

Array (element) indicating the base address of the source array, from which N elements are moved. Can be of any type except character.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable. See BLAS conventions in the introduction for interpretation of the LA<0 case.

B

Array (element) indicating the base address for the target array, to which N elements are moved. Must be of the same type as A.

LB

Stride for the B vector. Must be of type integer and not equal to zero. See BLAS conventions in the introduction for interpretation of the LB<0 case.

Externals

None

Dynamic Space

512 words if both LA and LB differ from 1.

Method

When both LA and LB equal unity, then a vector shift with zero shift count is used. Other cases are handled with the periodic gather or scatter instructions, or both.

NOTE

Periodic gather and scatter operations can be performed by this routine, but also by the three LIB99 routines VGATHP, VSCATP and VFILL.

Function SDOT
(from the BLAS package)

Purpose

To compute the dot product (inner product) between two vectors. See BLAS conventions in the introduction.

Format

DPROD=SDOT (N , A , LA , B , LB)

Parameters

DPROD

Function result of type real, containing the dot product of the two input vectors.

N

Number of elements in the dot product operation. Must be of type integer.

A

Array (element) indicating the base address of the first input operand. Must be of type real.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable. See BLAS conventions in the introduction for interpretation of the LA<0 case.

B

Array (element) indicating the base address of the second input operand. Must be of type real.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable. See BLAS conventions in the introduction for interpretation of the LB<0 case.

Externals

None

Dynamic Space

512 or 1024 words, depending on the values of LA and LB.

Method

Vector multiplication using the vector instruction MULFV, followed by vector summation using the vector instruction SUMFV.

Subroutine SITI64

Purpose

To convert one or several IEEE 16-bit integers, also known as Short Integers (INTEGER*2 in IBM notation), to Control Data 64-bit integer format. Both byte reversed (SITI64) and naturally ordered (I2TI64) integers are handled.

Format

```
CALL I2TI64( N , A , IXBYTA , B )
CALL SITI64( N , A , IXBYTA , B )
```

Parameters

N

Number of IEEE 16-bit integers to convert. Must be of type integer.

A

Input array holding the N contiguously stored IEEE 16-bit integers that are to be converted. Can be of any data type. The 2 bytes of each IEEE integer are assumed to be stored in natural order (01) when I2TI64 is called, and in reversed order (10) when SITI64 is called. The contents of A are not altered by this routine.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array A of the leftmost byte of the first integer to be converted. The beginning of A is represented by IXBYTA=0.

B

Output array in which the Control Data 64-bit integers are stored. Must be of type integer. Conversion in place is not allowed; that is, B and A must be distinct.

Externals

None

Dynamic Space

None

Remarks

An IEEE Short Integer is a 16-bit integer represented in 2's complement otation. A Control Data integer is also represented in 2's complement notation, but occupies 64 bits of storage.

No special type integers are recognized. In particular, the hexadecimal value Z"8000" (the most negative number representable using 16 bits) is converted to the Control Data 64-bit integer Z"FFFFFFFFFFFF8000" (decimal -32768), in spite of the fact that some hardware may attribute a special meaning to this number.

Function SNRM2

(from the BLAS package)

Purpose

To compute the Euclidean length, or 2-norm, of a vector of type real; that is, to compute the square root of the sum of the squares of its elements. See BLAS conventions in the introduction.

Format

ELEN=SNRM2 (N , A , LA)

Parameters

ELEN

Function result of type real, containing the square root of the sum of the squared elements of A.

N

Number of elements in the vector. Must be of type integer.

A

Array (element) indicating the base address of the input vector. Must be of type real.

LA

Stride for the A vector. Must be of type integer. Only the magnitude of LA is significant. Negative values are negated before use.

Externals

None

Dynamic Space

512 words.

Method

Vector multiplication (MULFV) to generate squares of the elements of A, followed by vector summation (SUMFV). The last step, to compute the square root of the computed sum, is handled by means of inline code identical to that of the system runtime library used by FORTRAN.

Subroutine SRTF64

Purpose

To convert one or several IEEE 32-bit floating-point numbers, also known as Short Real, to Control Data 64-bit floating-point format. Both byte reversed (SRTF64) and naturally ordered (SRNTF64) IEEE numbers are handled.

Format

```
CALL SRTF64 ( N , A , IXBYTA , B )
CALL SRNTF64( N , A , IXBYTA , B )
```

Parameters

N

Number of IEEE 32-bit floating-point numbers to convert. Must be of type integer.

A

Input array holding the N contiguously stored IEEE 32-bit numbers that are to be converted. Can be of any data type. The 4 bytes of each IEEE number are assumed to be stored in natural order (0123) when SRNTF64 is called, and in reversed order (3210) when SRTF64 is called. The contents of A are not altered by this routine.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array A of the leftmost byte of the first number to be converted. The beginning of A is represented by IXBYTA=0.

B

Output array in which the Control Data 64-bit floating-point numbers are stored. Must be of type real. Conversion in place is not allowed; that is, B and A must be distinct. See Remarks below for details about what happens when the IEEE number is an indefinite, an infinity, or when it is unnormalized.

Externals

None

Dynamic Space

None

Remarks

For a detailed description of the IEEE Short Real format, see the description of subroutine F64TSR. Special cases are treated as follows (ignoring the byte reversal assumed by SRTF64):

1. An IEEE signed indefinite is translated into a Control Data 64-bit signed indefinite. This routine (arbitrarily) chooses the hexadecimal number Z"7000000000000000" in the positive and Z"F000000000000000" in the negative case.

2. An IEEE signed infinity is translated into a Control Data 64-bit signed infinity. This routine (arbitrarily) chooses the hexadecimal number Z"5000000000000000" in the positive and Z"D000000000000000" in the negative case.

3. An unnormalized IEEE number gets converted and normalized.

Subroutine SSCAL
(from the BLAS package)

Purpose

To scale a real vector in place; that is, to multiply each of its elements by a real scalar constant. See BLAS conventions in the introduction.

Format

CALL SSCAL (N , S , A , LA)

Parameters

N

Number of elements of A to be scaled. Must be of type integer.

S

Scalar constant or variable or array element, which is used to scale the elements in the vector A. Must be of type real.

A

Array (element) indicating the base address of the vector to be scaled. Must be of type real.

LA

Stride for the A vector. Must be of type integer. Only the magnitude of LA is significant. Negative values are negated before use.

Externals

None

Dynamic Space

512 words if LA differs from 1.

Method

The vector instruction MULFV is used.

Subroutine SSWAP
(from the BLAS package)

Purpose

To exchange the elements of two vectors. See BLAS conventions in the introduction.

Format

CALL SSWAP (N , A , LA , B , LB)

Parameters

N

Number of elements to interchange between A and B. Must be of type integer.

A

Array (element) indicating the base address of the first vector. Can be of any type except character. This array must be distinct from B.

LA

Stride for the A vector. Must be of type integer and not equal to zero. See BLAS conventions in the introduction for interpretation of the LA<0 case.

B

Array (element) indicating the base address of the second vector. Must be of the same type as A. This array must be distinct from A.

LB

Stride for the B vector. Must be of type integer and not equal to zero. See BLAS conventions in the introduction for interpretation of the LB<0 case.

Externals

None

Dynamic Space

512 or 1024 words, depending on the values of LA and LB.

Method

Elements from one of the vectors are moved to a temporary space, using either a vector shift (SHFV) or a periodic gather (GTHV) instruction. This is followed by an analogous move from the other vector location to the first, using a temporary space if both LA and LB differ from unity. One more move completes the process.

Subroutine TRED2

Purpose

To reduce a real symmetric matrix to a real symmetric tridiagonal matrix. This is generally the first step in obtaining the complete eigensystem of the matrix.

Format

CALL TRED2 (N , A , KA , Z , KZ , D , S , T)

Parameters

N

The order of the input matrix A. Must be of type integer.

A

Array (element) indicating the address of the upper left-hand corner of the upper triangular part of the real symmetric matrix A, which is to be transformed to tridiagonal form. The contents of A will not be altered by TRED2 unless A is the same array as Z.

KA

Storage mode indicator for A. Must be of type integer. If A and Z are the same array, then KA=KZ must hold. This is possible only when KA.GE.N.

A value of 1 indicates that the columns of the upper triangular part of A are stored linearly, thus omitting the subdiagonal elements:

$$a_{00}\ a_{01}\ a_{11}\ a_{02}\ a_{12}\ a_{22}\ a_{03}\ \cdots$$

A value greater than or equal to N indicates full storage mode; that is, KA is the size of the first dimension (column length) of the two-dimensional array A. In this case the subdiagonal elements are ignored and will not be altered by this routine, unless A and Z are the same array which is permitted if KA=KZ.

Z

Array (element) indicating the address of the upper left-hand corner of a two-dimensional output array in which the orthogonal transformation matrix produced by the reduction will be stored. Must be of type real. If the contents of the matrix A do not need to be preserved (most eigenvalue solvers do not require A), Z may be the same array as A, provided that KA = KZ.

KZ

Size of the first dimension (column length) of array Z. Must be greater than or equal to N. Must be of type integer. If Z and A are identical, KZ = KA must hold.

D

Output array of length at least N words. Must be of type real. On return the diagonal elements of the produced tridiagonal matrix will be stored in D.

S

Output array of length at least N words. Must be of type real. On return S will contain (in its first N-1 positions) the superdiagonal elements of the produced symmetric tridiagonal matrix.

T

Scratch array of length at least N words. Can be of any type except character.

Parameters (Continued)	METHOD

This routine performs a Householder transformation, as presented in J. H. Wilkinson: The Algebraic Eigenvalue Problem, Oxford University Press, London, 233-236, 290-294 (1965). It closely resembles the implementation in the ALGOL procedure TRED2 in Wilkinson-Reinsch: Linear Algebra, Handbook for Automatic Computation, Vol.2, Springer-Verlag, Berlin, 212-226 (1971). The algorithm is very well-suited for vectorization, and the bulk of the work is done in vector mode--to a large extent utilizing the LIB99 routine SDOT to compute the dot product between two vectors.

Externals	LOCF, MXMOVU, RSMULV, RVAXPY, SCOPY, SDOT, SQRT, VFILL, VGATHP
Dynamic Space	None

Subroutine VABS

Purpose

To take the absolute values of the elements of a vector. Both integer and real types are supported.

Format

```
CALL IVABS ( N , A , LA , B , LB )
CALL RVABS ( N , A , LA , B , LB )
```

Parameters

N

Number of elements in the target vector B. Must be of type integer.

A

Array (element) indicating the base address of the source vector. Must be of type real if calling RVABS, and of type integer if calling IVABS.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the target vector. Must be of type real if calling RVABS, and of type integer if calling IVABS.

LB

Stride for the B vector. Must be of type integer and not equal to zero.

Externals

None

Dynamic Space

512 or 1024 words, depending on the values of LA and LB.

Method

In RVABS the vector instruction used is ANDV, where one of the operands is the broadcast scalar Z"7FFFFFFFFFFFFFFF". In IVABS the elements of A are first subtracted from zero using the vector instruction SUBXV. This vector is then selectively merged with the original elements, using the MRGV vector instruction controlled by the sign bits in the source vector.

Subroutine VADD

Purpose

To perform the vector operation C = A + B, where C is a vector and A and B are either scalars or vectors. Both integer and real types are supported.

Format

```
CALL ISADDV ( N , S ,  B , C )
CALL IVADD  ( N , A , LA , B , LB , C , LC )
CALL IVADDS ( N , A ,  S , C )
CALL IVADDV ( N , A ,  B , C )

CALL RSADDV ( N , S ,  B , C )
CALL RVADD  ( N , A , LA , B , LB , C , LC )
CALL RVADDS ( N , A ,  S , C )
CALL RVADDV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Must be of type real if RVADD, RVADDS, or RVADDV is called, and of type integer if IVADD, IVADDS, or IVADDV is called.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Must be of type real if RSADDV, RVADD, or RVADDV is called, and of type integer if ISADDV, IVADD, or IVADDV is called.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Must be of type real if RSADDV, RVADD, RVADDS or RVADDV is called, and of type integer if ISADDV, IVADD, IVADDS, or IVADDV is called.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Must be of type real if RSADDV or RVADDS is called, and of type integer if ISADDV or IVADDS is called.

Externals	None
Dynamic Space	512 words for each of the parameters LA, LB and LC which equal neither 1 nor 0.
Method	Vector addition using either the vector instruction ADDXV (integer operands) or ADDFV (real operands).
Remarks	Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters all arrays are assumed to be of this type.

Subroutine VAND

Purpose

To perform the logical 64-bit vector operation C = A .AND. B, where C is a vector and where A and B are either scalars or vectors.

Format

```
CALL SANDV ( N , S ,  B , C )
CALL VAND  ( N , A , LA , B , LB , C , LC )
CALL VANDS ( N , A ,  S , C )
CALL VANDV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Can be of any type except character.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Can be of any type except character.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Can be of any type except character.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Can be of any type except character.

Externals

None

Dynamic Space

0, 512, or 1024 words, depending on the values of LA, LB, and LC.

Method

The vector instruction ANDV is utilized.

Remarks

Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters all arrays are assumed to be of this type.

Subroutine VAXPY

Purpose

To compute a linked triad; that is, one of the two operations Z = A*X - Y (RVAXMY) and Z = A*X + Y (RVAXPY), where X, Y and Z are vectors and A is a scalar.

Format

```
CALL RVAXMY ( N , A , X , Y , Z )
CALL RVAXPY ( N , A , X , Y , Z )
```

Parameters

N

Number of elements in each of the three vectors. Must be of type integer.

A

Scalar constant or variable or array element. Must be of type real.

X

Array (element) indicating the base address of the first source vector. Must be of type real.

Y

Array (element) indicating the base address of the second source vector. Must be of type real.

Z

Array (element) indicating the base address of the target vector into which the result of the linked triad operation Z=A*X-Y (RVAXMY) or Z=A*X+Y (RVAXPY) is written. Must be of type real.

Externals

None

Dynamic Space

544 words.

Method

Vector multiplication of the scalar A by each of the elements in the vector X, using dynamic space as the target, followed by the subtraction (RVAXMY) or addition (RVAXPY) of the vector Y, putting the result into the vector Z.

NOTE

RVAXPY is very similar to the LIB99 subroutine SAXPY.

Subroutine VDIV

Purpose

To perform the vector operation C = A / B, where C is a vector and A and B are either scalars or vectors. Both integer and real types are supported.

Format

```
CALL ISDIVV ( N , S ,  B , C )
CALL IVDIV  ( N , A , LA , B , LB , C , LC )
CALL IVDIVS ( N , A ,  S , C )
CALL IVDIVV ( N , A ,  B , C )

CALL RSDIVV ( N , S ,  B , C )
CALL RVDIV  ( N , A , LA , B , LB , C , LC )
CALL RVDIVS ( N , A ,  S , C )
CALL RVDIVV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Must be of type real if RVDIV, RVDIVS or RVDIVV is used, and of type integer if IVDIV, IVDIVS, or IVDIVV is used.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Must be of type real if RSDIVV, RVDIV, or RVDIVV is used, and of type integer if ISDIVV, IVDIV, or IVDIVV is used.

LB

Stride for the B vector. Must be of type integer and not equal to zero. A value of 0 indicates that B is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Must be of type real if RSDIVV, RVDIV, RVDIVS or RVDIVV is used, and of type integer if ISDIVV, IVDIV, IVDIVS, or IVDIVV is used.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Must be of type real if RSDIVV or RVDIVS is called, and of type integer if ISDIVV or IVDIVS is called.

Externals	None
Dynamic Space	512 words for each of the parameters LA, LB and LC which equal neither 1 nor 0.
Method	The vector instruction DIVFV is used. Integer operands are first converted to floating-point format, using the vector instruction CNIFV. After DIVFV is used the reverse conversion is performed on the operands, using the vector instruction CNFIV.
Remarks	Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters all arrays are assumed to be of this type.

Subroutine VFILL

Purpose

To broadcast a scalar value into the given locations of a vector.

Format

CALL VFILL (N , S , A , LA)

Parameters

N

Number of locations to fill. Must be of type integer.

S

Scalar constant or variable or array element to be broadcast into A. Can be of any type except character.

A

Array (element) indicating the base address of the target vector to be filled. Must be of the same type as S.

LA

Stride for the A vector. Must be of type integer and not equal to zero.

Externals

None

Dynamic Space

None

Method

The periodic scatter instruction SCTV is used.

NOTE

The LIB99 subroutine SCOPY can perform the same task as VFILL.

Subroutine VFLOAT

Purpose

To convert the elements of a vector of type integer to type real.

Format

CALL VFLOAT (N , A , LA , B , LB)

Parameters

N

Number of elements to convert. Must be of type integer.

A

Array (element) indicating the base address of the source vector. Must be of type integer.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the target vector into which the resulting floating-point values are to be written. Must be of type real.

LB

Stride for the B vector. Must be of type integer and not equal to zero.

Externals

None

Dynamic Space

512 words if LB≠1.

Method

The vector instruction CNIFV is used.

Subroutine VFTF64

Purpose

To convert one or several VAX 32-bit floating-point numbers to Control Data 64-bit floating-point format.

Format

CALL VFTF64(N , A , IXBYTA , B)

Parameters

N

Number of VAX 32-bit floating-point numbers to convert. Must be of type integer.

A

Input array holding the N contiguously stored VAX 32-bit numbers that are to be converted. Can be of any data type. The 4 bytes of each VAX number are assumed to be stored in pairwise byte reversed order (1032 instead of 0123). The contents of A are not altered by this routine.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array A of the leftmost byte of the first number to be converted. The beginning of A is represented by IXBYTA=0.

B

Output array in which the Control Data 64-bit floating-point numbers are stored. Must be of type real. Conversion in place is not allowed; that is, B and A must be distinct. See Remarks below for details about what happens when the VAX number is an indefinite.

Externals

None

Dynamic Space

None

Remarks

For a detailed description of the VAX floating-point format, see the description of subroutine F64TVF.

The 32-bit quantity with the hexadecimal representation Z"80000000" is converted to a Control Data 64-bit positive indefinite, arbitrarily chosen as the hexadecimal number Z"7000000000000000".

Subroutine VGATHER

Purpose

To perform a random (VGATHR or VGATHR1) or periodic (VGATHP) gather operation; that is, to move elements from noncontiguous source to contiguous target locations. In the random cases the moves are controlled by an index list, while the periodic case is characterized by a constant stride offset between the gathered source elements. See the description of the INDEX parameter concerning the difference between VGATHR and VGATHR1.

Format

```
CALL VGATHP  ( N , A , STRIDE , B )
CALL VGATHR  ( N , A , INDEX  , B )
CALL VGATHR1 ( N , A , INDEX  , B )
```

Parameters

N

Number of elements to move from A to B. Must be of type integer.

A

Array (element) indicating the base address of the source array. Can be of any type except character.

INDEX

Integer index list containing pointers into array A. Must be of type integer. This index list is used only by VGATHR and VGATHR1. If INDEX(J)=K, the element A(K) is moved to the location B(J). When VGATHR is called, it is assumed that the base address A is referenced by the subscript 0; that is, A and A(0) represent the same address. In contrast, when VGATHR1 is called, the first element of A is assumed to be referenced as A(1), using the subscript 1 rather than 0.

STRIDE

Constant stride for the A array, used only by VGATHP. Must be of type integer and not equal to zero. Defines the elements to be moved as (A(J*STRIDE),J=0,...,N-1).

B

Array (element) indicating the base address of target array. Must be of the same type as A.

Externals

None

Dynamic Space

None

Method

In both VGATHR and VGATHR1, a scalar loop unrolled to four levels is used. VGATHP does the work by means of the periodic gather instruction GTHV.

NOTE

The LIB99 subroutine SCOPY can also perform periodic gather operations.

Subroutine VIFIX

Purpose	To convert the elements of a vector of type real to type integer.
Format	CALL VIFIX (N , A , LA , B , LB)
Parameters	N
	Number of elements to convert. Must be of type integer.
	A
	Array (element) indicating the base address of the source vector. Must be of type real.
	LA
	Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.
	B
	Array (element) indicating the base address of the target vector into which the resulting integer values are to be written. Must be of type integer.
	LB
	Stride for the B vector. Must be of type integer and not equal to zero.
Externals	None
Dynamic Space	512 words if LB≠1.
Method	The vector instruction CNFIV is used.

Subroutine VIOR

Purpose

To perform the logical 64-bit vector operation C = A .OR. B, where C is a vector and A and B are either scalars or vectors.

Format

```
CALL SIORV ( N , S ,  B , C )
CALL VIOR  ( N , A , LA , B , LB , C , LC )
CALL VIORS ( N , A ,  S , C)
CALL VIORV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Can be of any type except character.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Can be of any type except character.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable.

C

Array (element) indicating the base address of the target vector, into which the result of the vector operation is written. Can be of any type except character.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Can be of any type.

Externals

None

Dynamic Space

0, 512, or 1024 words, depending on the values of LA, LB, and LC.

Method

The vector instruction IORV is utilized.

Remarks

Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters all arrays are assumed to be of this type.

Subroutine VMASK

Purpose

To create a vector by means of selecting elements from two source vectors, letting the choice between the two be determined by whether the corresponding element in a logical input vector has the value .TRUE. or .FALSE..

Format

```
CALL VMASK0 ( N , A0 , LA0 , B  ,  LB , A1 , LA1 )
CALL VMASK1 ( N , A1 , LA1 , B  ,  LB , A0 , LA0 )
CALL VMASK2 ( N , A1 , LA1 , A0 , LA0 , B  , C  , LC )
```

To determine which call to use, consult the table in the description of parameter B.

Parameters

N

Number of elements in each vector. Must be of type integer.

A1

Array (element) indicating the base address of the vector whose elements are to be chosen whenever the corresponding element in B has the value .TRUE.; that is, when its leftmost bit is set (=1). Can be of any type except character.

LA1

Stride for the A1 vector. Must be of type integer. Must not be zero if VMASK0 is used. A value of 0 indicates that A1 is a scalar variable or constant.

A0

Array (element) indicating the base address of the vector whose elements are to be chosen whenever the corresponding element in B has the value .FALSE.; that is, when its leftmost bit is cleared (=0). Must be of the same type as A1.

LA0

Stride for the A0 vector. Must be of type integer. Must not be zero if VMASK1 is used. A value of 0 indicates that A0 is a scalar variable or constant.

B

Array (element) indicating the base address of the control vector. The type of array B should in principle be logical. However, the vector instruction actually used for the selective moves really only cares about how the leftmost bit looks in a given element of B. For example, a negative integer or floating-point value is interpreted as .TRUE. because the sign bit is set (leftmost bit=1).

The contents of B control the selection of elements from A1 and A0 in the following manner:

	VMASK0	VMASK1	VMASK2
B(K)=.TRUE.	no action	A0(K)=A1(K)	C(K)=A1(K)
B(K)=.FALSE.	A1(K)=A0(K)	no action	C(K)=A0(K)

Parameters (Continued)	LB Stride for the B vector. Used only in calls to VMASK0 and VMASK1. Must be of type integer and not equal to zero. C Array (element) indicating the base address of the target vector in VMASK2. Must be of the same type as A1 and A0. LC Stride for the C vector. Must be of type integer and not equal to zero.
Externals	None
Dynamic Space	0 - 1536 words, depending on the values of LA0, LA1, LB and LC.
Method	Vector moves, using a mixture of the vector instructions SHFV, SCTV, GTHV, XORV and MRGV.

Subroutine VMASUM

Purpose

To compute the moving absolute sum of the elements of a vector. A window of length NW is moved along the input vector, one step at a time. For each position, the output element whose location corresponds to the center of the window is computed as the sum of the absolute values of the elements enclosed by the window.

Format

CALL VMASUM (N , NW , A , B)

Parameters

N

Length of the input vector A and the output vector B. Must be of type integer.

NW

Length of the summing window. Must be of type integer. The value of NW should be odd and not exceed N. When even it will be treated as if it had the value NW+1.

A

Array (element) indicating the base address of the input vector. Must be of type real.

B

Array (element) indicating the base address of the target vector. Must be of type real. The arrays A and B must be distinct. B is filled with the moving absolute sums computed from the vector A. More precisely, the element B(K), where K=0,1,...,N-1, is computed as the sum of the absolute values of the elements A(K-NW/2) through A(K+NW/2), where A(J) is considered to be equal to zero whenever J<0 or J>N-1.

Externals

None

Dynamic Space

(N + NW/2) words.

Method

A completely vectorized method is used.

Remarks

Whenever the window is positioned so that a part of it extends outside vector A, the corresponding element of B is computed as the sum of fewer than NW absolute values. Thus, in effect, the window size increases uniformly from 1+NW/2 to NW in the beginning, and decreases the same way in the end.

Subroutine VMUL

Purpose

To perform the vector operation C = A * B, where C is a vector, and where A and B are either scalars or vectors. Both integer and real types are supported.

Format

```
CALL ISMULV ( N ,  S ,  B ,  C )
CALL IVMUL  ( N ,  A , LA ,  B , LB , C , LC )
CALL IVMULS ( N ,  A ,  S ,  C )
CALL IVMULV ( N ,  A ,  B ,  C )

CALL RSMULV ( N ,  S ,  B ,  C )
CALL RVMUL  ( N ,  A , LA ,  B , LB , C , LC )
CALL RVMULS ( N ,  A ,  S ,  C )
CALL RVMULV ( N ,  A ,  B ,  C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Must be of type real if RVMUL, RVMULS, or RVMULV is used, and of type integer if IVMUL, IVMULS, or IVMULV is used.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Must be of type real if RSMULV, RVMUL, or RVMULV is used or of type integer if ISMULV, IVMUL, or IVMULV is used.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Must be of type real if RSMULV, RVMUL, RVMULS, or RVMULV is used, and of type integer if ISMULV, IVMUL, IVMULS, or IVMULV is used.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Must be of type real if RSMULV or RVMULS is called and of type integer if ISMULV or IVMULS is called.

Externals	None
Dynamic Space	512 words for each of the parameters LA, LB and LC which equal neither 1 nor 0.
Method	The vector instruction MULFV is used. Integer operands are first converted to floating-point format, using the vector instruction CNIFV. After MULFV is used, the reverse conversion is performed on the operands using the vector instruction CNFIV.
Remarks	Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters, all arrays are assumed to be of this type.

Subroutine VRANF

Purpose	To fill all locations in a vector with random numbers.
Format	CALL VRANF (N , A , LA)
Parameters	N
	Number of random numbers to insert into A. Must be of type integer.
	A
	Array (element) indicating the base address of the target vector into which the random numbers are written. Must be of type real.
	LA
	Stride for the A vector. Must be of type integer and not equal to zero.
Externals	None
Dynamic Space	None
Method	A scalar loop with repeated references to the system provided function RANF.

Subroutine VSCATTER

Purpose

To perform a random (VSCATR or VSCATR1) or periodic (VSCATP) scatter operation; that is, to move elements from contiguous source to noncontiguous target locations. In the random cases the moves are controlled by an index list, while the periodic case is characterized by a constant stride offset between the target locations scattered into. See the description of the index parameter concerning the difference between VSCATR and VSCATR1.

Format

```
CALL VSCATP  ( N , A , STRIDE , B )
CALL VSCATR  ( N , A , INDEX  , B )
CALL VSCATR1 ( N , A , INDEX  , B )
```

Parameters

N

Number of elements to move from A to B. Must be of type integer.

A

Array (element) indicating the base address of the source vector. Can be of any type except character.

INDEX

Integer index list containing pointers into array B. Must be of type integer. This index list is used only by VSCATR and VSCATR1. If INDEX(J)=K this is interpreted as a request to move the element A(J) to the location B(K). When VSCATR is called, it is assumed that the base address B is referenced by the subscript 0; that is, B and B(0) represent the same address. In contrast, when VSCATR1 is called, the first element of B is assumed to be referenced as B(1), using the subscript 1 rather than 0.

STRIDE

Constant stride for the B vector, used only by VSCATP. Must be of type integer and not equal to zero. Defines the target locations as (B(J*STRIDE),J=0,...,N-1).

B

Array (element) indicating the base address of the target array. Must be of the same type as A.

Externals

None

Dynamic Space

None

Method

In both VSCATR and VSCATR1 a scalar loop unrolled to four levels is used. VSCATP does the work by means of the periodic scatter instruction SCTV.

Remark

The best way to scatter a scalar is by a call to the LIB99 subroutine VFILL.

NOTE

The LIB99 subroutines SCOPY and VFILL can also perform periodic scatter operations.

Subroutine VSHFC

Purpose

To perform the circular vector shift operation C = SHIFT(A , K), where C is a vector and A and K are either scalars or vectors. Note that the integer shift count vector K may contain either positive or negative integers, but that the corresponding left and right shifts, respectively, are both circular.

Format

```
CALL SSHFCV ( N , S ,  K , C )
CALL VSHFC  ( N , A , LA , K , LK , C , LC )
CALL VSHFCS ( N , A ,  J , C )
CALL VSHFCV ( N , A ,  K , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the source vector whose elements are to be shifted, as determined by the contents of K (or J). Can be of any type except character.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

K

Array (element) indicating the base address of the second vector operand, containing the positive or negative integer shift counts. Must be of type integer.

LK

Stride for the K vector. Must be of type integer. A value of 0 indicates that K is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the shifted elements of A are written. Can be of any type except character.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

J

Scalar shift count to be used for all elements of A when VSHFC is called. Must be of type integer.

S

Scalar constant or variable or array element to be broadcast as the first input operand when SSHFCV is called. Can be of any type except character.

Externals	None
Dynamic Space	0, 512, or 1024 words, depending on the values of LA, LB, and LC.
Method	The vector instruction SHFV is used.
Remarks	Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters all arrays are assumed to be of this type.

Subroutine VSUB

Purpose

To perform the vector operation C = A - B, where C is a vector and A and B are either scalars or vectors. Both integer and real types are supported.

Format

```
CALL ISSUBV ( N , S ,  B , C )
CALL IVSUB  ( N , A , LA , B , LB , C , LC )
CALL IVSUBS ( N , A ,  S , C )
call IVSUBV ( N , A ,  B , C )

CALL RSSUBV ( N , S ,  B , C )
CALL RVSUB  ( N , A , LA , B , LB , C , LC )
CALL RVSUBS ( N , A ,  S , C )
CALL RVSUBV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Must be of type real if RVSUB, RVSUBS, or RVSUBV is called, and of type integer if IVSUB, IVSUBS, or IVSUBV is called.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Must be of type real if RSSUBV, RVSUB, or RVSUBV is called, and of type integer if ISSUBV, IVSUB, or IVSUBV is called.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Must be of type real if RSSUBV, RVSUB, RVSUBS, or RVSUBV is called, and of type integer if ISSUBV, IVSUB, IVSUBS, or IVSUBV is called.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Must be of type real if RSSUBV or RVSUBS is called, and of type integer if ISSUBV or IVSUBS is called.

Externals	None
Dynamic Space	512 words for each of the parameters LA, LB and LC which equal neither 1 nor 0.
Method	Vector subtraction using either the vector instruction SUBXV (integer operands) or SUBFV (real operands).
Remarks	Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters, all arrays are assumed to be of this type.

Function VSUM

Purpose

To compute the sum of either the algebraic (RVSUM) or absolute (RVASUM) values of the elements of a real vector.

Format

```
SUM=RVASUM ( N , A , LA )
SUM=RVSUM  ( N , A , LA )
```

Parameters

SUM

Function result of type real. If RVASUM is called, the result represents the sum of the absolute values of the elements of A. If RVSUM is called, the result represents the sum of the algebraic (unaltered) values of the elements of A.

N

Number of elements to be summed. Must be of type integer.

A

Array (element) indicating the base address of the input vector. Must be of type real.

LA

Stride for the A vector. Must be of type integer and not equal to zero.

Externals

None

Dynamic Space

512 words.

Method

The vector instruction SUMFV is used to compute the element sum. In RVASUM the vector instruction ANDV is used prior to the summing in order to obtain the absolute values of the elements in A.

NOTE

RVASUM is very similar to the LIB99 function SASUM.

Subroutine VXOR

Purpose

To perform the logical 64-bit vector operation C = A .XOR. B, where C is a vector and A and B are either scalars or vectors.

Format

```
CALL SXORV ( N , S ,  B , C )
CALL VXOR  ( N , A , LA , B , LB , C , LC )
CALL VXORS ( N , A ,  S , C )
CALL VXORV ( N , A ,  B , C )
```

Parameters

N

Number of elements in each vector. Must be of type integer.

A

Array (element) indicating the base address of the first vector operand. Can be of any type except character.

LA

Stride for the A vector. Must be of type integer. A value of 0 indicates that A is a scalar variable or constant.

B

Array (element) indicating the base address of the second vector operand. Can be of any type except character.

LB

Stride for the B vector. Must be of type integer. A value of 0 indicates that B is a scalar variable or constant.

C

Array (element) indicating the base address of the target vector into which the result of the vector operation is written. Can be of any type except character.

LC

Stride for the C vector. Must be of type integer and not equal to zero.

S

Scalar constant or variable or array element to be broadcast as either the first or second input operand in the vector operation. Can be of any type except character.

Externals

None

Dynamic Space

0, 512, or 1024 words, depending on the values of LA, LB, and LC.

Method

The vector instruction XORV is used.

Remarks

Vectors with contiguously stored elements are characterized by stride values of 1. In routines which do not accept stride parameters, all arrays are assumed to be of this type.

Subroutine WITI64

Purpose

To convert one or several IEEE 32-bit integers, also known as Word Integers (INTEGER*4 in IBM notation), to Control Data 64-bit integer format. Both byte reversed (WITI64) and naturally ordered (I4TI64) integers are handled.

Format

```
CALL I4TI64( N , A , IXBYTA , B )
CALL WITI64( N , A , IXBYTA , B )
```

Parameters

N

Number of IEEE 32-bit integers to convert. Must be of type integer.

A

Input array holding the N contiguously stored IEEE 32-bit integers that are to be converted. Can be of any data type. The 4 bytes of each IEEE integer are assumed to be stored in natural order (0123) when I4TI64 is called, and in reversed order (3210) when WITI64 is called. The contents of A are not altered by this routine.

IXBYTA

FORTRAN-type subscript, or item count, in units of 8-bit bytes. Must be a nonnegative integer. Specifies the location in array A of the leftmost byte of the first integer to be converted. The beginning of A is represented by IXBYTA=0.

B

Output array in which the Control Data 64-bit integers are stored. Must be of type integer. Conversion in place is not allowed; that is, B and A must be distinct.

Externals

None

Dynamic Space

None

Remarks

An IEEE Word Integer is a 32-bit integer represented in 2's complement notation. A Control Data integer is also represented in 2's complement notation, but occupies 64 bits of storage.

No special type integers are recognized. In particular, the hexadecimal value Z"80000000" (the most negative number representable using 32 bits) is converted to the Control Data 64-bit integer Z"FFFFFFFF80000000" (decimal -134217728), in spite of the fact that some hardware may attribute a special meaning to this number.

Related Manuals A

The manuals that are referenced in this manual and those that contain background information to this manual are listed in table A-1.

A complete list of NOS/VE manuals is given in the SCL Language Definition manual. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
/EXPLAIN
```

Table A-1. Related Manuals

Manual Title	Publication Number	Online Manual Title
FORTRAN for NOS/VE Language Definition Usage	60485913	
FORTRAN for NOS/VE Quick Reference		FORTRAN
FORTRAN Version 2 for NOS/VE Language Definition Usage	60487113	
FORTRAN Version 2 for NOS/VE Quick Reference		VFORTRAN
Math Library for NOS/VE Usage	60486413	
SCL for NOS/VE Language Definition Usage	60464013	
SCL for NOS/VE Source Code Management Usage	60464313	
SCL for NOS/VE Object Code Management Usage	60464413	

Ordering Printed Manuals

Control Data manuals are available through Control Data sales offices or through:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103
(612) 292-2101

When ordering a manual, please specify the complete title, publication number, and revision level.

Accessing Online Manuals

To access an online manual, log in to NOS/VE and supply the online title (listed in table A-1) on the EXPLAIN command. For example, to see the FORTRAN Quick Reference online manual, enter

```
EXPLAIN MANUAL=FORTRAN
```

Index

Q

R

S

T

V

W

FORTRAN for NOS/VE LIB99 Usage, 60485915 C

We would like your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who Are You?

___ Manager
___ Systems Analyst or Programmer
___ Applications Programmer
___ Operator
___ Other _____

How Do You Use This Manual?

___ As an Overview
___ To learn the Product/System
___ For Comprehensive Reference
___ For Quick Look-up

Do You Also Have?

___ FORTRAN for NOS/VE Language Definition Usage
___ FORTRAN for NOS/VE Tutorial
___ FORTRAN for NOS/VE Quick Reference
___ FORTRAN for NOS/VE Summary

What programming languages do you use? _____

How Do You Like This Manual? Check those that apply.

Yes	Somewhat	No	
___	___	___	Is the manual easy to read (print size, page layout, and so on)?
___	___	___	Is it easy to understand?
___	___	___	Is the order of topics logical?
___	___	___	Are there enough examples?
___	___	___	Are the examples helpful? (___ Too simple ___ Too complex)
___	___	___	Is the technical information accurate?
___	___	___	Can you easily find what you want?
___	___	___	Do the illustrations help you?
___	___	___	Does the manual tell you what you need to know about the topic?

Comments? If applicable, note page number and paragraph.

Would you like a reply? ___ Yes ___ No

Continue on other side

From:

Name _____ Company _____

Address _____ Date _____

_____ Phone No. _____

_____

Please send program listing and output if applicable to your comment.

'APE

TAPE

)LD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Technology and Publications Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

CUT ALONG LINE

)LD

FOLD


CONTROL DATA