AMT

DAP Series

General Support Library

GSLIB

(man010.02)

AMT endeavours to ensure that the information in this document is correct, but does not accept responsibility for any error or omission.

Any procedure described in this document for operating AMT equipment should be read and understood by the operator before the equipment is used. To ensure that AMT equipment functions without risk to safety or health, such procedures should be strictly observed by the operator.

The development of AMT products and services is continuous and published information may not be up to date. Any particular issue of a product may contain part only of the facilities described in this document or may contain facilities not described here. It is important to check the current position with AMT.

Specifications and statements as to performance in this document are AMT estimates intended for general guidance. They may require adjustment in particular circumstances and are therefore not formal offers or undertakings.

Statements in this document are not part of a contract or program product licence save in so far as they are incorporated into a contract or licence by express reference. Issue of this document does not entitle the recipient to access to or use of the products described, and such access or use may be subject to separate contracts or licences.

Technical publication man010.02

First edition 20 October 1987
Second edition 22 April 1988

AMT will be pleased to receive readers' views on the contents, organisation, etc of this publication. Please make contact at either of the addresses below:

Publications Manager
Active Memory Technology Ltd
65 Suttons Park Avenue
Reading
Berks, RG6 1AZ, UK

Tel: 0734 661111

Publications Manager
Active Memory Technology Inc
16802 Aston St Suite 103
Irvine
California, 92714, USA

Tel: (714) 261 8901

Contents

CONTENTS

Chapter 1

Introduction

1.1 Background

The General Support subroutine library was developed at Queen Mary College (QMC) in London and is jointly owned by AMT and QMC. The library is a set of 93 routines which can be called from FORTRAN-PLUS. The contents of the library are based on those of the DAP Fortran library at QMC, which grew in response to user requests for specific routines. The routines were provided by members of the DAP Support Unit (DAPSU) at QMC, or were written at the suggestion of DAPSU members, or were submitted by users themselves. Many of the algorithms used by these routines have been in regular use on a first generation DAP at QMC since 1980.

1.2 Arrangement of Documentation

The routines described in this manual are classified by chapter, arranged in a NAG-like manner, covering such areas as solution of linear equations, Fourier transforms, and so on. The next chapter in this manual provides a full listing of the contents of the library, chapter by chapter, and gives a brief description of the area covered by each routine.

1.3 Validation

Before being added to the library all routines undergo validation tests, designed and written at DAPSU. These tests have been collected together in a validation suite, which is used to check installation of the library.

1.4 Full-form Documentation

The full description of each routine has eleven sections, covering the following areas:

1 Purpose
2 Specification
3 Description
4 References
5 Arguments
6 Error Indicators
7 Auxiliary Routines
8 Accuracy
9 Further Comments
10 Keywords
11 Example

1.4.1 Purpose

The purpose of the routine is given, and where relevant, details of the area covered by the routine.

1.4.2 Specification

The calling sequence to be used when you invoke the routine. If the routine is written in FORTRAN-PLUS, **Specification** gives the declaration statements at the head of the routine; if the routine is written in APAL, the equivalent statements are given.

1.4.3 Description

The description of the algorithm used by the routine is given.

1.4.4 References

Any references used in connection with the routine are given.

1.4.5 Arguments

The significance of each argument used by the routine is explained.

1.4.6 Error Indicators

The significance of any error indicators returned by the routine is explained.

1.4.7 Auxiliary routines

The names of any auxiliary routines used by the routine are given. The auxiliary routines are kept in the same library as the subroutine library routines but are not, in general, available to users.

1.4.8 Accuracy

Some indication is given of the expected accuracy of any result returned by the routine as a result of the method used to calculate it. No information is given about results with respect to the word length used; for such information have a look at the routines in chapter 12 (X02 – Machine constants).

1.4.9 Further Comments

Any information which does not fall under any other heading is included here.

1.4.10 Keywords

This section is intended for use with an information retrieval system and gives a list of subjects to which the operation of the routine may be relevant.

1.4.11 Example

An example program is given (both Host and DAP programs) for each of the routines, showing the use of the routine and any expected results.

WARNING

You should follow closely the specification of the calling sequence given in section 2 of the details of each routine in the following chapters, otherwise you may get unexpected results.

1.5 Access to the Library

The subroutine library is linked in at the consolidation stage of the compiling process. For more details than are included below, see the relevant AMT publication: *Program Development Under UNIX* (man003), or *Program Development Under VAX/VMS* (man004).

1.5.1 Using the library under UNIX

The library resides within the UNIX system as:

```
/usr/lib/dap/sulib.dl
```

and you can use it in a call to **dapa** or **dapf** by means of the **-l** flag, as in:

```
dapf -o myfile.dd myfile.df -l sulib
```

This call will compile the DAP section **myfile.df**, linking in any routines from the library and produce a DOF file **myfile.dd**.

1.5.2 Using the library under VAX/VMS

The library resides within the VMS system as:

```
SYS$LIBRARY:GSLIB.DLB
```

and you can use it in a call to **DLINK** using the **/LIBRARY** qualifier, as in:

```
$ DLINK MYFILE,SYS$LIBRARY:GSLIB/LIBRARY
```

This call links the DAP object code in file **MYFILE.DOB** with any library routines you might specify in your source code, producing an executable DAP program in file **MYFILE.DEX**.

Alternatively, you can use the **DAP_LIBRARY** logical name, as in:

```
$ DEFINE DAP_LIBRARY SYS$LIBRARY:GSLIB
```

This call will cause the library to be searched automatically in all subsequent **DLINK** operations. If you use the library frequently, you may find it convenient to include the above line in your **LOGIN.COM** file.. If there are several DAP users on your system, your system manager could include the line:

```
$ DEFINE/SYSTEM DAP_LIBRARY SYS$LIBRARY:GSLIB
```

in the system startup command file, to give all users automatic access to the library.

1.6 Other AMT subroutine libraries

This General Support subroutine library forms one of a series of libraries available from AMT. Other libraries include:

- Low level graphics library
- Signal processing library
- Image Processing library

details of which can be obtained from your local AMT representative.

Chapter 2

GSLIB quick-reference catalogue

Listed below are the groups of subroutines in release 1 of GSLIB, the General Support subroutine library, and the subroutines in each group; each group is allocated a chapter in this manual. Release 1 of the library is targetted at the DAP 500 series of machines, those with an edge size of 32.

You may find this chapter helpful in the initial selection of suitable routines for the job in hand.

Chapter 3: A03 – Variable precision arithmetic

1 **A03_ADD_PLANES_I1** adds bit planes together by performing an addition of n consecutive bits under each processing element. It returns the result of this addition as an INTEGER*1 MATRIX. Any overflow past bit 7 is discarded and the result is given modulo 128.

Chapter 4: C06 – Summation of series, including fast Fourier transformations

1 **C06_LFT_LV** performs a one dimensional finite Fourier transform of 1024 complex points.

2 **C06_LFT_ESS** calculates the two dimensional discrete Fourier transform of 32^2 complex points.

Chapter 5: F01 – Matrix operations, including inversion

1 **F01_G_MM** performs a general matrix multiply of two matrices A and B where A is a P by Q matrix and B is a Q by R matrix with P, Q and R in the range 1 to 32.

2 **F01_M_INV** calculates, in place, the inverse of a given N by N matrix with N in the range 1 to 32.

3 **F01_MM_STRASSEN** uses Strassen's algorithm to multiply two (partitioned) 64^2 matrices.

Chapter 6: F02 – Eigenvalues and eigenvectors

1 **F02_ALL_EIG_VALS_TD_LV** finds all the eigenvalues of a symmetric tridiagonal matrix of order up to 1024 using Sturm sequences.

2 **F02_ALL_EIG_VALS_TD_ES** finds all the eigenvalues of a symmetric tridiagonal matrix of order up to 32 using Sturm sequences.

3 **F02_EIG_VALS_TD_LV** finds up to 32 selected eigenvalues of a symmetric tridiagonal matrix of order up to 1024 using Sturm sequences.

4 **F02_JACOBI** calculates the eigenvalues and eigenvectors of a real symmetric matrix. The method is based on the classical Jacobi algorithm using plane rotations.

Chapter 7: F04 – Simultaneous linear equations

1 **F04_BIGSOLVE** solves large sets of linear equations. The maximum size of the system depends on the size of the DAP store. The matrix of the coefficients of the equations is of size **SIZE** by **SIZE** and the right hand side is assumed to be held in column **SIZE+1**. The whole matrix is held in the DAP partitioned in **DAPSIZE** blocks. This routine is not recommended for systems of order 32 or less – in this case, you should use the routine F04_GJN_LE_ES.

2 **F04_GJ_NLE_ES** solves for x the system of linear equations $Ax = b$, where A is a non-sparse matrix of order N (in the range 1 to 32), using the Gauss Jordan method.

3 **F04_QR_GIVENS_SOLVE** solves for x the linear system $Ax = b$, where A is an N by N matrix with $2 < N < 33$. The routine may be used to solve up to 32 different right hand side vectors b simultaneously.

4 **F04_TRIDS_ES** returns the solution of a tridiagonal linear system of equations of order up to 32. It finds vector x, where:

$$Mx = y$$

and M is a tridiagonal matrix.

5 **F04_TRIDS_ES_SQ** returns the solution of a set of up to 32 tridiagonal linear systems of equations each of order up to 32. It solves up to 32 systems of the form:

$$Mx = y$$

where M is a tridiagonal matrix.

6 **F04_TRIDS_LV** returns the solution of a tridiagonal linear system of equations of order up to 1024. It finds vector x, where:

$$Mx = y$$

and M is a tridiagonal matrix.

Chapter 8: G05 – Random numbers

1 **G05_MC_BEGIN** sets the basic generator routine Z_G05_MC_INT to an initial state.

2 **G05_MC_I4** returns an INTEGER*4 MATRIX containing 1024 pseudo-random integer numbers taken from a uniform distribution between 0 and $2^{31} - 1$.

3 **G05_MC_I8** returns an INTEGER*8 MATRIX containing 1024 pseudo-random integer numbers taken from a uniform distribution between 1 and $2^{59} - 1$.

4 **G05_MC_NORMAL_R4** returns a REAL*4 MATRIX of 1024 normal pseudo-random variates from the distribution $N(0,1)$.

5 **G05_MC_R4** returns a REAL*4 MATRIX of 1024 pseudo-random real numbers taken from a uniform distribution between 0 and 1.

6 **G05_MC_R8** returns a REAL*8 MATRIX of 1024 pseudo-random real numbers taken from a uniform distribution between 0 and 1.

7 **G05_MC_REPEAT** sets the basic generator routine Z_G05_MC_INT to a repeatable initial state.

Chapter 9: H – Operations research, graph structures, networks

1 **H01_L_ASSIGN** solves the linear assignment problem with a minimum objective function and a real cost matrix of order N by N, where N <= 32.

Chapter 10: J06 – Plotting

1 **J06_CHAR_CONT** returns a character matrix containing a rough contour map of a real matrix. You can control the number of contours and contour levels.

2 **J06_ZEBRA_CHART** returns a contour map of a real matrix suitable for output to a printing device. The output is called a ZEBRA chart as it consists of alternating bands of blanks and a given character.

Chapter 11: M01 – Sorting

1 **M01_BSORT_LV** is based on bitonic sorting. Data is sorted according to a key, or the key alone may be sorted.

2 **M01_INV_PERMUTE_COLS** permutes the first M columns of a matrix according to a permutation vector (IV). The routine is equivalent to the FORTRAN-PLUS statements:

```
      DO 10 I = 1, M
   10 A_PERMUTED(,IV(I)) = A(,I)
```

3 **M01_INV_PERMUTE_LV_32** permutes the values in an INTEGER*4 or REAL*4 matrix using an INTEGER*4 matrix key. The result is written to a new matrix and the original data is unaffected. The data shuffling implemented is ANSWER (KEY(I)) = START (I), for I = 1, 1024, using long vector indexing. Hence the key matrix must contain values in the range 1 – 1024, but the values need not be distinct.

4 **M01_INV_PERMUTE_ROWS** permutes the first M rows of a matrix according to a permutation vector (IV). The routine is equivalent to the FORTRAN-PLUS statements:

```
   DO 10 I = 1, M
10 A_PERMUTED(,IV(I) ) = A(,I )
```

5 **M01_PERMUTE_COLS** permutes the first M columns of a matrix according to a permutation vector (IV). The routine is equivalent to the FORTRAN-PLUS statements:

```
   DO 10 I = 1, M
10 A_PERMUTED(,I ) = A (,IV(I))
```

6 **M01_PERMUTE_LV_32** permutes the values in an INTEGER*4 or REAL*4 matrix using an INTEGER*4 matrix key. The result is written to a new matrix and the original data is unaffected. The data shuffling implemented is ANSWER (I) = START (KEY(I)), for I = 1,1024, using long vector indexing. Hence the key matrix must contain values in the range 1 – 1024, but the values need not be distinct.

7 **M01_PERMUTE_ROWS** permutes the first M rows of a matrix according to a permutation vector (IV). The result is equivalent to the FORTRAN-PLUS statements:

```
   DO 10 I = 1, M
10 A_PERMUTED(I,) = A (IV(I),)
```

8 **M01_SORT_V_I 4** sorts the first N elements of an integer vector into ascending or descending order. The permutation required to perform the sort is returned to the calling routine.

9 **M01_SORT_V_R4** sorts the first N elements of a real vector into ascending or descending order. The permutation required to perform the sort is returned to the calling routine.

Chapter 12: S – Special functions

1 **S04_ARC_COS** returns the value of the inverse cosine function $\arccos(x)$ for a matrix argument. The result lies in the range $[0, \pi]$.

2 **S04_ARC_SIN** returns the value of the inverse sine function $\arcsin(x)$ for a matrix argument. The result lies in the range $[-\pi/2, \pi/2]$.

3 **S04_ATAN2_M** is a matrix function similar to the standard FORTRAN ATAN2 function. It calculates arc-tangent(matrix-1/matrix-2), and returns a matrix of values in the range $-\pi$ to π, in the correct quadrant, and with divide-by-zero errors avoided. If a zero divided by zero is attempted then a zero is returned.

4 **S04_ATAN2_V** is a vector function similar to the standard FORTRAN ATAN2 function. It calculates arc-tangent(vector-1/vector-2), and returns a vector of values in the range $-\pi$ to π, in the correct quadrant, and with divide-by-zero errors avoided. If a zero divided by zero is attempted then a zero is returned.

5 **S04_COS_INT** returns the value of the cosine integral $C_i x$ for a matrix argument.

6 **S04_MOD_BES_I0** returns the value of the modified Bessel function I 0 for a matrix argument.

7 **S04_MOD_BES_I1** returns the value of the modified Bessel function I 1 for a matrix argument.

8 **S04_SIN_INT** returns the value of the sine integral $S_i x$ for a matrix argument.

9 **S15_ERF** returns the value of the error function.

10 **S15_ERFC** returns the value of the complement of the error function.

Chapter 13: X01 – Mathematical constants

1 **X01_PI** determines the value of π for any of the real precision lengths available on the DAP.

Chapter 14: X02 – Machine constants

1 **X02_EPSILON** determines the smallest positive real (EPS) such that 1.0+EPS differs from 1.0, for any of the real precision lengths available on the DAP.

2 **X02_MAXDEC** determines the value of MAXDEC for the different precision lengths available on the DAP. MAXDEC is the maximum number of decimal digits which can be represented accurately over the whole range of floating point numbers.

3 **X02_MAXINT** determines the value of MAXINT for the different precision lengths available on the DAP. MAXINT is the largest integer such that MAXINT and –MAXINT can both be represented accuratetly.

4 **X02_MAXPW2** determines the value of MAXPW2 for the different precision lengths available on the DAP. MAXPW2 is the largest integer power to which 2.0 may be raised without overflow.

5 **X02_MINPW2** determines the value of MINPW2 for the different precision lengths available on the DAP. MINPW2 is the largest negative integer power to which 2.0 may be raised without underflow.

6 **X02_RMAX** determines the largest real (RMAX) such that RMAX and –RMAX can both be represented exactly, for any of the real precision lengths available on the DAP.

7 **X02_RMIN** determines the smallest real (RMIN) such that RMIN and –RMIN can both be represented exactly, for any of the real precision lengths available on the DAP.

8 **X02_TOL** determines the value of TOL (= RMIN/EPSILON) for any of the precision lengths available on the DAP.

Chapter 15: X05 – Other utilities

1 **X05_ALT_LV** produces a long vector of alternating groups of N false values followed by N true values and so on, until all components of the vector have a value. If the value of N lies outside the range 1 to 1024 all components will have the value false.

2 **X05_CRINKLE** effects a transformation in data storage format for vertical mode data occupying an array of matrices – from 'sliced' to 'crinkled' storage.

3 **X05_EAST_BOUNDARY** returns a logical matrix containing at most one .TRUE. in each row corresponding to the last .TRUE. (if any) in each row of the logical matrix parameter. The routine is equivalent to the FORTRAN-PLUS code:

```
      DO 10 I = 1, 32
      IF (.NOT. ANY (LM (I, ))) GOTO 10
      KM (I, ) = REV (FRST (REV (LM (I, ))))
   10 CONTINUE
```

4 **X05_E_MAX_PC** returns a logical matrix whose i^{th} row has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} row of the real matrix argument holding the maximum value in that row, and .FALSE. elsewhere.

5 **X05_E_MAX_PR** returns a logical matrix whose i^{th} column has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} column of the real matrix argument holding the maximum value in that column, and .FALSE. elsewhere.

6 **X05_E_MAX_VC** returns a real vector whose i^{th} component is the maximum value in the i^{th} row of the real matrix argument.

7 **X05_E_MAX_VR** returns a real vector whose i^{th} component is the maximum value in the i^{th} column of the real matrix argument.

8 **X05_E_MIN_PC** returns a logical matrix whose i^{th} row has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} row of the real matrix argument holding the minimum value in that row, and .FALSE. elsewhere.

9 **X05_E_MIN_PR** returns a logical matrix whose i^{th} column has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} column of the real matrix argument holding the minimum value in that column, and .FALSE. elsewhere.

10 **X05_E_MIN_VC** returns a real vector whose i^{th} component is the minimum value in the i^{th} row of the real matrix argument.

11 **X05_E_MIN_VR** returns a real vector whose i^{th} component is the minimum value in the i^{th} column of the real matrix argument.

12 **X05_EXCH_P** exchanges L planes starting at X with L planes starting at Y under activity control indicated by M. The planes are exchanged in increasing order; you are cautioned about the strange effects which will occur if the two sets of planes overlap.

13 **X05_GATHER_V_32** assigns to the components of a vector the values of those components of a vector array designated by corresponding components of an indexing vector. The index values are interpreted as reduced rank indices to the vector array.

14 **X05_I_MAX_PC** returns a logical matrix whose i^{th} row has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} row of the integer matrix argument holding the maximum value in that row, and .FALSE. elsewhere.

15 **X05_I_MAX_PR** returns a logical matrix whose i^{th} column has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} column of the integer matrix argument holding the maximum value in that column, and .FALSE. elsewhere.

16 **X05_I_MAX_VC** returns an integer vector whose i^{th} component is the maximum value in the i^{th} row of the integer matrix argument.

17 **X05_I_MAX_VR** returns an integer vector whose i^{th} component is the maximum value in the i^{th} column of the integer matrix argument.

18 **X05_I_MIN_PC** returns a logical matrix whose i^{th} row has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} row of the integer matrix argument holding the minimum value in that row, and .FALSE. elsewhere.

19 **X05_I_MIN_PR** returns a logical matrix whose i^{th} column has the value .TRUE. in the position(s) corresponding to the position(s) in the i^{th} column of the integer matrix argument holding the minimum value in that column, and .FALSE. elsewhere.

20 **X05_I_MIN_VC** returns an integer vector whose i^{th} component is the minimum value in the i^{th} row of the integer matrix argument.

21 **X05_I_MIN_VR** returns an integer vector whose i^{th} component is the minimum value in the i^{th} column of the integer matrix argument.

22 **X05_LOG2** returns the value:

$$[\log(N-1)]+1$$

where square brackets indicate 'integer part of', and N is the input argument. The routine returns the number of steps required in a $\log_2$, recursive doubling, algorithm.

23 **X05_LONG_INDEX** generates an integer matrix whose i^{th} element in long vector order is $(i + N - 1)$, where N is a parameter to the routine.

24 **X05_NORTH_BOUNDARY** returns a logical matrix containing at most one .TRUE. in each column corresponding to the first .TRUE. (if any) in each column of the logical matrix parameter. The routine is equivalent to the FORTRAN-PLUS code:

```
      DO 10 I = 1, 32
      IF (.NOT.ANY(LM(,I))) GOTO 10
      KM(,I) = FRST(LM(,I))
   10 CONTINUE
```

25 **X05_PATTERN** produces four user-selectable patterns, each of which is returned as a logical matrix. The four patterns available are:

0 – The main diagonal

1 – The minor diagonal

2 – A matrix, the rows of which correspond to the rows generated by ALTC

3 – The unit lower triangular matrix

26 **X05_SCATTER_V_32** takes components of a vector and assigns the values to components of a vector array designated by corresponding components of an indexing vector. The index values are interpreted as reduced rank indices to the vector array.

27 **X05_SHLC_LV** performs a cyclic long vector shift to the left on up to 128 bit planes.

28 **X05_SHLP_LV** performs a planar long vector shift to the left on up to 128 bit planes.

29 **X05_SHORT_INDEX** generates an integer vector whose i^{th} element is $(i + \mathrm{N} - 1)$, where N is a parameter to the routine.

30 **X05_SHRC_LV** performs a cyclic long vector shift to the right on up to 128 bit planes.

31 **X05_SHRP_LV** performs a planar long vector shift to the right on up to 128 bit planes.

32 **X05_SOUTH_BOUNDARY** returns a logical matrix containing at most one .TRUE. in each column corresponding to the last .TRUE. (if any) in each column of the logical matrix parameter. The routine is equivalent to the FORTRAN-PLUS code:

```
    DO 10 I = 1, 32
    IF (.NOT. ANY (LM (,I))) GOTO 10
    KM (,I) = REV (FRST (REV (LM (,I))))
10  CONTINUE
```

33 **X05_STRETCH_4** stretches the first quarter of a real matrix A (considered as a long vector), such that each element is repeated four times consecutively.

34 **X05_STRETCH_8** stretches the first eighth of a real matrix A (considered as a long vector), such that each element is repeated eight times consecutively.

35 **X05_STRETCH_N** stretches the first $\mathrm{N^{th}}$ of a real matrix A (considered as a long vector), such that each element is repeated N times consecutively, N being 2 raised to a positive integer power.

36 **X05_SUM_LEFT_I2** takes as input the long vector A (an INTEGER*2 vector) and returns an INTEGER*2 long vector each of whose elements is the sum of all the elements on the left of, but not including, the corresponding element of A.

37 **X05_SUM_RIGHT_I2** takes as input the long vector A (an INTEGER*2 vector) and returns an INTEGER*2 long vector each of whose elements is the sum of all the elements on the right of , but not including, the corresponding element of A.

38 **X05_UNCRINKLE** effects a transformation in data storage format for vertical mode data occupying an array of matrices – from 'crinkled' to 'sliced' storage.

39 **X05_WEST_BOUNDARY** returns a logical matrix containing at most one .TRUE. in each row corresponding to the first .TRUE. (if any) in each row of the logical matrix parameter. The routine is equivalent to the FORTRAN-PLUS code:

```
      DO 10 I = 1, 32
      IF (.NOT.ANY(LM,(I,))) GOTO 10
      KM(I,) = FRST(LM(I,))
   10 CONTINUE
```

Chapter 3

A03 – Variable precision arithmetic

Contents:

3.1 A03_ADD_PLANES_I1

release 1

1 Purpose

A03_ADD_PLANES_I1 adds bit planes together, that is, it performs an addition of n consecutive bits of each PE.

A03_ADD_PLANES_I1 returns the result of this addition so that the corresponding element of the result is the sum of the n consecutive bits of the corresponding PE.

The result is calculated to an accuracy of integer*1, therefore any overflow past bit 7 is thrown away and the result is modulo 128.

2 Specification

```
   INTEGER*1 MATRIX FUNCTION A03_ADD_PLANES_I1 (STARTPLANE ,
+     NRPLANES)
   INTEGER NRPLANES
   <any type> STARTPLANE(,)
```

3 Description

The DAP can add the contents of a store plane and the Q and C planes simultaneously ; this routine uses that ability to add pairs of planes. The resulting carry is then rippled up the answer.

4 References

None

5 Arguments

STARTPLANE – <*any type*> MATRIX

On entry *STARTPLANE* contains the address of the first plane to be added. The function adds *NRPLANES* consecutive planes starting at *STARTPLANE*. *STARTPLANE* may, in FORTRAN-PLUS, be any variable represented by a plane address. None of the planes added are changed by the function, but you are warned against allowing the destination of the result to overlap the planes to be added. If you do try overlapping the planes, the program will still work, but you will have overwritten your arguments before you accessed them!

NRPLANES – INTEGER

On entry *NRPLANES* specifies the number of planes to be added. Unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

The results are calculated mod 128 – overflow is not detected.

9 Further Comments

None

10 Keywords

Bit summation, integer addition.

11 Example

The example adds the bit planes which define a long index vector, thus counting the number of bits set .TRUE. in the binary representation of the integers 0 to 1023.

Host program

```
      PROGRAM MAIN

      INTEGER IM(1024)
      COMMON /IM/IM

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('IM',IM,1024)

      WRITE(6,1000)
 1000 FORMAT(6X,'I',3X,'No. of bits set'//)
      DO 10 II=1,1024
      I=II-1
 10   WRITE(6,2000) I,IM(II)
 2000 FORMAT(I7,10X,I2)

      CALL DAPREL
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      INTEGER*1 IM1(,)
      INTEGER IM(,)
      LOGICAL LM(,,32)
      COMMON /IM/IM

      EQUIVALENCE (IM,LM)

      EXTERNAL INTEGER*1 MATRIX FUNCTION A03_ADD_PLANES_I1
```

```
      CALL XO5LONGINDEX(IM,0)
      IM1=A03_ADD_PLANES_I1(LM(,,21),10)
      IM=IM1
      CALL CONVMFI(IM)

      RETURN
      END
```

Results

```
      I   No. of bits set


      0          0
      1          1
      2          1
      3          2
      .          .
      .          .
      .          .

   1020          8
   1021          9
   1022          9
   1023         10
```

Chapter 4

C06 – Summation of series

(including fast fourier transformations)

Contents:

4.1 C06_FFT_ESS

release 1

1 Purpose

C06_FFT_ESS calculates the two dimensional discrete Fourier transform of 32 x 32 complex points.

2 Specification

```
SUBROUTINE C06_FFT_ESS (X , Y , INVERS , FIRST)
REAL X (,) , Y (,)
LOGICAL INVERS , FIRST
```

3 Description

The 2D transform is calculated by performing independent sets of row and column 32-point transforms.

The data is then in bit reversed order independently in rows and columns and a final shuffle is performed to reorder the data.

For a description of the general theory of FFTs see [1].

4 References

[1] BRIGHAM E.O.

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – REAL MATRIX

On entry X contains the real part of the data to be transformed. On exit X contains the real part of the transformed data.

Y – REAL MATRIX

On entry Y contains the imaginary part of the data to be transformed. On exit Y contains the imaginary part of the transformed data.

INVERS – LOGICAL

If INVERS is set to .FALSE. the transform:

$$X_{jk}+iY_{jk} = \sum_m \sum_n (A_{mn}+iB_{mn}) exp\left(2\pi i\frac{(j-1)(m-1)}{32} + \frac{(k-1)(n-1)}{32}\right)$$

is calculated, where $j = 1, 2, \ldots, 32$; $k = 1, 2, \ldots, 32$ and the summations are also over $m = 1, 2, \ldots, 32$ and $n = 1, 2, \ldots, 32$; and where $i = \sqrt{-1}$.

If INVERS is set to .TRUE. the transform:

$$A_{mn}+iB_{mn} = \sum_j \sum_k (X_{jk}+Y_{jk}) exp\left(-2\pi i\frac{(m-1)(j-1)}{32} + \frac{(n-1)(k-1)}{32}\right)$$

is calculated, where $m = 1, 2, \ldots, 32$; $n = 1, 2, \ldots, 32$ and the summations are also over $j = 1, 2, \ldots, 32$ and $k = 1, 2, \ldots, 32$; and where $i = \sqrt{-1}$.

FIRST – LOGICAL

If FIRST is set to .TRUE. the exponential coefficients for the transform are calculated. Consequently FIRST must be set to .TRUE. the first time this routine is called within a program, but may be set to .FALSE. for all subsequent calls.

6 Error Indicators

None

7 Auxiliary Routines

This routine calls the DAP library routines Z_C06_F2DCOEFF, Z_C06_ROWFFT, Z_C06_COLFFT and Z_C06_F2DBREV.

8 Accuracy

Accuracy will be data dependent. Some indication of the accuracy may be obtained by performing a subsequent inverse transform and comparing the results with the original data.

9 Further Comments

This routine uses a common block with the name CC06FFTESSQ. Consequently the user program must not use a common block with this name.

10 Keywords

Fast Fourier Transform

11 Example

The example given sets up an initial array of complex points in which the real and imaginary parts are simple functions of a real variable. A forward transform is then performed followed by a back transform of the transformed data. The first 32 complex values of the first row of the initial data, transformed data and back transformed data are printed.

Host program

```
      PROGRAM HTFFTESS
      REAL X(32,32),Y(32,32),XT(32,32),YT(32,32),XB(32,32),YB(32,32)
      COMMON /BDATA/X,Y,XT,YT,XB,YB
      CALL dapcon('tfftess.dd')
      CALL dapent('TFFTESS')
      CALL daprec('BDATA',X,6*1024)
      DO 100 i=1,1
      WRITE(6,6001)
      WRITE(6,6002)
     $(X(J,i),Y(J,I),XT(J,I),YT(J,I),XB(J,I),YB(J,I),J=1,32)
 6001 FORMAT(2X,'DATA TO BE TRANSFORMED',9X,'TRANSFORMED DATA'
     $9X,'BACK TRANSFORMED DATA'//3(9X,'REAL', 9X,'IMAG') /)
 6002 FORMAT(6(1X,F12.6))
 100  CONTINUE
      CALL daprel
      STOP
      END
```

DAP program

```
ENTRY SUBROUTINE TFFTESS
REAL X(,),Y(,),XT(,),YT(,),XB(,),YB(,)
INTEGER IM(,)
LOGICAL INVERS,FIRST
COMMON /BDATA/X,Y,XT,YT,XB,YB
CALL LONG_INDEX(IM)
X=6.28318*(IM-1)/1023.0
Y=SIN(X)
X=COS(X)*COS(X)
XT=X
YT=Y
INVERS=.FALSE.
FIRST=.TRUE.
CALL C06_FFT_ESS(XT,YT,INVERS,FIRST)
XB=XT
YB=YT
FIRST=.FALSE.
INVERS=.TRUE.
CALL C06_FFT_ESS(XB,YB,INVERS,FIRST)
XB=XB/1024.0
YB=YB/1024.0
CALL CONVMFE(X)
CALL CONVMFE(Y)
CALL CONVMFE(XT)
CALL CONVMFE(YT)
CALL CONVMFE(XB)
CALL CONVMFE(YB)
RETURN
END
```

Results

DATA TO BE TRANSFORMED		TRANSFORMED DATA		BACK TRANSFORMED DATA	
REAL	IMAG	REAL	IMAG	REAL	IMAG
1.000000	.000000	512.499512	-.000001	1.000000	.000000
.999962	.006142	.029227	-.002885	.999962	.006142
.999848	.012284	.014964	-.002954	.999849	.012284
.999661	.018425	.009909	-.002994	.999661	.018425
.999397	.024565	.007302	-.003027	.999397	.024565
.999057	.030705	.005657	-.002998	.999057	.030705
.998642	.036843	.004522	-.003013	.998642	.036843
.998152	.042980	.003741	-.003100	.998152	.042980
.997588	.049116	.003037	-.003032	.997588	.049115
.996947	.055249	.002486	-.003049	.996948	.055249
.996232	.061381	.002015	-.003077	.996232	.061380
.995442	.067510	.001615	-.003032	.995441	.067510
.994578	.073636	.001249	-.003026	.994577	.073636
.993638	.079760	.000901	-.003026	.993638	.079760
.992624	.085881	.000625	-.003057	.992624	.085881
.991536	.091999	.000311	-.003093	.991536	.091999
.990374	.098113	.000000	-.003080	.990374	.098113
.989138	.104223	-.000266	-.003058	.989137	.104223
.987827	.110329	-.000591	-.003060	.987828	.110329
.986444	.116432	-.000956	-.003115	.986444	.116432
.984986	.122530	-.001285	-.003113	.984986	.122530
.983456	.128623	-.001659	-.003107	.983457	.128623
.981852	.134711	-.002083	-.003071	.981853	.134711
.980176	.140795	-.002545	-.003089	.980176	.140795
.978428	.146873	-.003098	-.003093	.978428	.146873
.976608	.152945	-.003736	-.003047	.976608	.152945
.974715	.159012	-.004658	-.003119	.974715	.159012
.972751	.165073	-.005815	-.003132	.972751	.165073
.970715	.171127	-.007510	-.003113	.970715	.171127
.968608	.177175	-.010330	-.003156	.968608	.177175
.966431	.183217	-.015892	-.003190	.966431	.183217
.964184	.189251	-.033177	-.003261	.964183	.189251

4.2 C06_FFT_LV

release 1

1 Purpose

C06_FFT_LV performs a one dimensional finite Fourier transform of 1024 complex points.

2 Specification

```
SUBROUTINE C06_FFT_LV (X , Y , INVERS , FIRST)
REAL X(,) , Y(,)
LOGICAL INVERS , FIRST
```

3 Description

The data is considered as 1024 complex points in long vector order, and the transform is calculated by performing linked row and column transforms. The first step is to calculate 32-point transforms along each row of complex data. The results of the row transforms are multiplied by a second set of exponential factors and then 32-point transforms are calculated along each column in a similar way to the row transforms but using different exponential factors. The exponential factors are set up in such a way as to ensure that the row and column transforms are linked correctly to give the required 1D transform. The final step re-orders the data which is in bit reversed order.

For a description of the general theory of FFTs see [1].

4 References

[1] BRIGHAM E.O.

The Fast Fourier Transform: Prentice-Hall, 1974

5 Arguments

X – REAL MATRIX

On entry X contains the real part of the data to be transformed. On exit X contains the transformed real part of the data.

Y – REAL MATRIX

On entry Y contains the imaginary part of the data to be transformed. On exit Y contains the transformed imaginary part of the data.

INVERS - LOGICAL

If INVERS is set to .FALSE. the transform:

$$X_j+iY_j = \sum_{k+1}^{1024}(A_k+iB_k)exp\left(2\pi i\frac{(j-1)(k-1)}{1024}\right)$$

is calculated, where j = 1, 2, ... , 1024 and the summation is over k = 1, 2, ... , 1024; and where $i = \sqrt{-1}$.

If INVERS is set to .TRUE. the transform:

$$A_k+iB_k = \sum_{j+1}^{1024}(X_j+iY_j)exp\left(-2\pi i\frac{(j-1)(k-1)}{1024}\right)$$

is calculated, where k = 1, 2, ... , 1024 and the summation is over j = 1, 2, ... , 1024; and where $i = \sqrt{-1}$.

The argument is unchanged on exit.

FIRST – LOGICAL

If FIRST is set to .TRUE. the exponential coefficients for the transform are calculated. Consequently FIRST must be set to .TRUE. the first time this routine is called within a program,but may be set to .FALSE. for all subsequent calls.

The argument is unchanged on exit.

6 Error Indicator

None

7 Auxiliary Routines

The routine calls the DAP library routines Z_C06FFT1DCOEFF, Z_C06ROWFFT, Z_C06COLFFT, Z_C06FFT1DBREV.

8 Accuracy

Accuracy will be data dependent. You can get some idea of the accuracy by carrying out the transform, then carrying out the inverse transform and comparing the results with the original data.

9 Further Comments

The routine uses a common block with name CC06FFTLV. Consequently your program must not use a common block with this name.

10 Keywords

Fast Fourier Transform

11 Example

The example given sets up initial data in which the real and imaginary parts are simple functions of a real variable. A forward transform is then performed, followed by a back transform of the transformed data. The first ten complex values of the initial data, transformed data and back transformed data are printed in long vector order.

Host program

```
      PROGRAM HTFFTLV
      REAL X(32,32),Y(32,32),XT(32,32),YT(32,32),XB(32,32),YB(32,32)
      COMMON /BDATA/X,Y,XT,YT,XB,YB

      CALL DAPCON('tfftlv.dd')
      CALL DAPENT('TFFTLV')
      CALL DAPREC('BDATA',X,6*1024)
      WRITE(6,6001)
      WRITE(6,6002) (X(I,1),Y(I,1),I=1,10)
      WRITE(6,6003)
      WRITE(6,6002) (XT(I,1),YT(I,1),I=1,10)
      WRITE(6,6004)
      WRITE(6,6002) (XB(I,1),YB(I,1),I=1,10)
6001  FORMAT(2X,'DATA TO BE TRANSFORMED'//7X,'REAL',9X,'IMAG'/)
6002  FORMAT(2(1X,F12.6))
6003  FORMAT(//2X,'TRANSFORMED DATA'//7X,'REAL',9X,'IMAG'/)
6004  FORMAT(//2X,'BACK TRANSFORMED DATA'//7X,'REAL',9X,'IMAG')
      STOP
      END
```

DAP Program

```
      ENTRY SUBROUTINE TFFTLV
      REAL X(,),Y(,),XT(,),YT(,),XB(,),YB(,)
      INTEGER IM(,)
      LOGICAL INVERS,FIRST
      COMMON /BDATA/X,Y,XT,YT,XB,YB
      CALL LONG_INDEX(IM)
      X=6.28318*(IM-1)/1023.0
      Y=SIN(X)
      X=COS(X)*COS(X)
      XT=X
      YT=Y
      INVERS=.FALSE.
      FIRST=.TRUE.
      CALL C06_FFT_LV(XT,YT,INVERS,FIRST)
      XB=XT
      YB=YT
      FIRST=.FALSE.
      INVERS=.TRUE.
      CALL C06_FFT_LV(XB,YB,INVERS,FIRST)
      XB=XB/1024.0
      YB=YB/1024.0
```

```
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      CALL CONVMFE(XT)
      CALL CONVMFE(YT)
      CALL CONVMFE(XB)
      CALL CONVMFE(YB)
      RETURN
      END
```

Results

```
      DATA TO BE TRANSFORMED

       REAL          IMAG

     1.000000       .000000
      .999962       .006142
      .999848       .012284
      .999661       .018425
      .999397       .024565
      .999057       .030705
      .998642       .036843
      .998152       .042980
      .997588       .049116
      .996947       .055249


  TRANSFORMED DATA

       REAL          IMAG

   512.499512      -.000001
  -511.081055      1.567145
   256.785889     -1.574793
     -.025975       .000161
      .099694      -.001184
      .113036      -.001728
      .108699      -.002016
      .101061      -.002178
      .093657      -.002353
      .086534      -.002373
```

```
BACK TRANSFORMED DATA

     REAL            IMAG
    .999999         .000000
    .999961         .006134
    .999847         .012277
    .999661         .018417
    .999397         .024560
    .999058         .030699
    .998641         .036837
    .998152         .042973
    .997588         .049111
    .996948         .055240
```

Chapter 5

F01 – Matrix Operations

(including inversion)

Contents:

5.1 F01_G_MM

release 1

1 Purpose

F01_G_MM performs a general matrix multiply of two matrices A and B, where A is a P by Q matrix and B is a Q by R matrix, with P, Q and R in the range 1 to 32.

2 Specification

```
REAL MATRIX FUNCTION F01_G_MM(A , B , P , Q , R , IFAIL)
REAL A(,) , B(,)
INTEGER P , Q , R , IFAIL
```

3 Description

The routine is an optimised general matrix multiply using one of the following three procedures, depending on the relative sizes of P,Q and R (see [1]).

Procedure 1

```
    F01_G_MM = 0.0
    DO 10 I = 1, Q
10  F01_G_MM = F01_G_MM + MATC(A(,I))*MATR(B(I,))
```

Procedure 2

```
    DO 10 I = 1, P
10  F01_G_MM(I,) = SUMR(MATC(A(I,))*B)
```

Procedure 3

```
    DO 10 I=1,R
10  F01_G_MM(,I)=SUMC(A*MATR(B(,I)))
```

If P/Q > 0.75 and R/Q > 0.75 procedure 1 is used, otherwise if P >= R procedure 3 is used or if P< R procedure 2 is used; the number 0.75 was determined empirically.

4 References

[1] MCKEOWN J J

Multiplication of non-standard matrices on DAP: DAP newsletter no 7: available from the DAP Suppoprt Unit, Queen Mary College, Mile End Road, London E1 4NS

5 Arguments

A – REAL MATRIX

On entry A contains the first of the two matrices to be multiplied together - array elements outside the matrix to be multiplied must be set to zero. The contents of A are unchanged on exit.

B – REAL MATRIX

On entry B contains the second of the two matrices to be multiplied together - array elements outside the matrix to be multiplied must be set to zero. The contents of B are unchanged on exit.

P – INTEGER

The number of rows in the first matrix. Unchanged on exit.

Q – INTEGER

The number of columns in the first matrix and the number of rows in the second matrix. Unchanged on exit.

R – INTEGER

The number of columns in the second matrix. Unchanged on exit.

IFAIL - INTEGER

Unless the routine detects an error (see **Error indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1

At least one of P, Q or R is not in the range 1 to 32.

7 Auxiliary Routines

None

8 Accuracy

You can expect six significant figures.

9 Further Comments

None

10 Keywords

Matrix multiply.

11 Example

The example given multiplies a 3 by 5 matrix of 1s by a 5 by 4 matrix of 1s.

Host program

```
      PROGRAM HTGMM
      INTEGER P,Q,R
      REAL A(32,32),B(32,32),C(32,32)
      COMMON /BN/P,Q,R
      COMMON /BIFAIL/IFAIL
      COMMON /BDATA/A,B,C
      READ(5,*) P,Q,R
      CALL dapcon('tgmm.dd')
      CALL dapsen('BN',p,3)
      CALL dapent('TGMM')
```

```
      CALL daprec('BDATA',A,3*1024)
      WRITE(6,6000) IFAIL
      WRITE(6,6001) ((A(I,J),J=1,6),I=1,6)
      WRITE(6,6002)
      WRITE(6,6001) ((B(I,J),J=1,6),I=1,6)
      WRITE(6,6002)
      WRITE(6,6001) ((C(I,J),J=1,6),I=1,6)
 6000 FORMAT(3X,I1//)
 6001 FORMAT(6(1X,F5.2)/)
 6002 FORMAT(/)
      CALL DAPREL
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE TGMM
      REAL A(,),B(,),C(,)
      INTEGER P,Q,R
      COMMON /BN/P,Q,R
      COMMON /BIFAIL/IFAIL
      COMMON /BDATA/A,B,C
      EXTERNAL REAL MATRIX FUNCTION F01_G_MM
      CALL CONVFSI(P,3)
      A=0.0
      B=0.0
      A(ROWS(1,P).AND.COLS(1,Q))=1.0
      B(ROWS(1,Q).AND.COLS(1,R))=1.0
      C=0.0
      C=F01_G_MM(A,B,P,Q,R,IFAIL)
      CALL CONVMFE(A)
      CALL CONVMFE(B)
      CALL CONVMFE(C)
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Data

```
  3  5  4
```

Results

```
 0

 1.00   1.00   1.00   1.00   1.00   0.00
 1.00   1.00   1.00   1.00   1.00   0.00
 1.00   1.00   1.00   1.00   1.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00

 1.00   1.00   1.00   1.00   0.00   0.00
 1.00   1.00   1.00   1.00   0.00   0.00
 1.00   1.00   1.00   1.00   0.00   0.00
 1.00   1.00   1.00   1.00   0.00   0.00
 1.00   1.00   1.00   1.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00

 5.00   5.00   5.00   5.00   0.00   0.00
 5.00   5.00   5.00   5.00   0.00   0.00
 5.00   5.00   5.00   5.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00
 0.00   0.00   0.00   0.00   0.00   0.00
```

5.2 F01_M_INV

release 1

1 Purpose

F01_M_INV calculates,in place, the inverse of a given N by N matrix with N in the range 1 to 32.

2 Specification

```
SUBROUTINE F01_M_INV (A , N , IFAIL)
REAL A (,)
INTEGER N , IFAIL
```

3 Description

The matrix is inverted using Gauss-Jordan elimination with full pivoting.

4 References

None

5 Arguments

A – REAL MATRIX

On entry A contains the matrix to be inverted, which is assumed to be located in the top left of A and array elements outside the input matrix must be set to zero. On exit A contains the inverse of that matrix.

N – INTEGER

On entry N must be set to the order of the matrix to be inverted. N is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N is not in the range 1 to 32.
IFAIL = 2	A pivot element is equal to zero – the matrix is singular.

7 Auxiliary Routines

None

8 Accuracy

You can expect five or six significant figures for well conditioned problems.

9 Further Comments

None

10 Keywords

Matrix inversion, Gauss-Jordan elimination.

11 Example

The example given inverts an N by N matrix, with N = 5 in this case. The matrix is generated as pseudo-random numbers in the range 0.0, 1.0, ... , 9.0 and then the diagonal elements are set to the sum of the elements in each row, thus ensuring a diagonally dominant, and so well conditioned matrix. The inverse matrix is multiplied by the original matrix as a check.

The results consist of the original matrix, the inverse matrix and their product.

Host program

```
      PROGRAM HTMINV
      REAL A(32,32),B(32,32),C(32,32)
      COMMON /BN/N
      COMMON /BDATA/A,B,C
      COMMON /BIFAIL/IFAIL
      READ(5,*) N
      CALL dapcon('tmin.dd')
      CALL DAPSEN('BN',N,1)
      CALL DAPENT('TMINV')
      CALL DAPREC('BDATA',A,3*1024)
      CALL DAPREC('BIFAIL',IFAIL,1)
      WRITE(6,6000) IFAIL
      WRITE(6,6001) ((A(I,J),J=1,5),I=1,5)
      WRITE(6,6002)
      WRITE(6,6001) ((B(I,J),J=1,5),I=1,5)
      WRITE(6,6002)
      WRITE(6,6001) ((C(I,J),J=1,5),I=1,5)
6000  FORMAT(2X,I2)
6001  FORMAT(5(2X,F10.6))
6002  FORMAT(/)
      CALL DAPREL
      STOP
      END
```

DAP progam

```
      ENTRY SUBROUTINE TMINV
C
      REAL A(,),B(,),C(,)
      INTEGER IM(,)
      COMMON /BN/N
      COMMON /BDATA/A,B,C
      COMMON /BIFAIL/IFAIL
      EXTERNAL REAL MATRIX FUNCTION G05MCR4
      EXTERNAL LOGICAL MATRIX FUNCTION X05PATTERN
      EXTERNAL REAL MATRIX FUNCTION F01GMM
      CALL CONVFSI(N,1)
```

```
      CALL G05MCBEGIN
      IM=10.0*G05MCR4(X)
      A=0.0
      A(ROWS(1,N).AND.COLS(1,N))=IM
      A(X05PATTERN(0))=MATC(SUMC(A(,)))
      B=A
C
      CALL F01_M_INV(B,N,IFAIL)
C
      C=0.0
      C=F01_G_MM(A,B,N,N,N,IERR)
C
      CALL CONVMFE(A)
      CALL CONVMFE(B)
      CALL CONVMFE(C)
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Data

```
5
```

Results

```
0


35.000000    8.000000    3.000000    8.000000    8.000000
 2.000000   21.000000    7.000000    3.000000    5.000000
 4.000000    1.000000   19.000000    4.000000    5.000000
 4.000000    6.000000     .000000   25.000000    9.000000
 6.000000    9.000000    1.000000    7.000000   32.000000


  .030777    -.007744    -.001791    -.007485    -.004099
  .000214     .050931    -.018557    -.001932    -.004569
 -.004507     .003413     .052401    -.005672    -.005999
 -.003178    -.006852     .003615     .044393    -.011185
 -.004995    -.011480     .003127    -.007587     .035938

 1.000000     .000000     .000000     .000000     .000000
  .000000     .999999     .000000     .000000     .000000
  .000000     .000000     .999999     .000000     .000000
  .000000     .000000     .000000    1.000001     .000000
  .000000     .000000     .000000     .000000    1.000000
```

5.3 F01_MM_STRASSEN

release 1

1 Purpose

F01_MM_STRASSEN uses Strassen's algorithm to multiply two (partitioned) 64 by 64 matrices.

2 Specification

```
SUBROUTINE F01_MM_STRASSEN (A , B , C)
REAL A (, , 2, 2) , B (, , 2, 2) , C (, , 2, 2)
```

3 Description

There is a well known result due to Strassen showing that 2 by 2 matrices may be multiplied using seven multiplications and fifteen additions instead of the eight multiplications and four additions required by the 'normal' method. This result is applied to the multiplication of 64 by 64 matrices partitioned into 2 by 2 sub-matrices of size 32 by 32. [1].

4 References

[1] PARKINSON D

Some interesting and useful results from complexity theory: DAP Newsletter no 2, p 8, August 1979: available from the DAP Support Unit, Queen Mary College, Mile End Road, London E1 4NS

5 Arguments

A – REAL MATRIX array of dimension (, , 2, 2)

On exit the 64 by 64 elements of the matrix set A contain the values of the matrix product

B – REAL MATRIX array of dimension (, , 2, 2)

Before entry the elements of B must be set to the first of the 64 by 64 matrices to be multiplied. Unchanged on exit.

C – REAL MATRIX array of dimension (, , 2, 2)

Before entry the elements of C must be set to the second of the 64 by 64 matrices to be multiplied. Unchanged on exit. All the matrices must be partitioned into four equal sub-matrices.

11	12
21	22

The matrix (, , I, J) is occupied by the data area shown as I J above.

6 Error Indicators

None

7 Auxiliary Routines

This routine calls the DAP library routine Z_F01_MM_N.

8 Accuracy

Depends on the data; you can normally expect six significant figures.

9 Further Comments

None

10 Keywords

Matrix multiplication, partitioned matrices, Strassen's algorithm.

11 Example

Host program

```
      PROGRAM STRASSENTEST

      REAL A(32,32),B(32,32),D,E
      LOGICAL FLAG

      COMMON/TEST/A,B
      COMMON/FLAG/FLAG

      DO 1 J = 1,32
        DO 1 I = 1,32
          D = I
          E = J
          A(I,J) = D*E - 2.
           B(I,J) = (D + E)*3.
 1    CONTINUE
      CALL dapcon('testmult.dd')
      CALL dapsen('TEST',A,2*1024)
      CALL dapent('TESTMULT')
      CALL daprec('FLAG',FLAG,1)
      CALL daprel
      IF(.NOT.FLAG) GO TO 2
      WRITE(6,100)
 100  FORMAT(20X,37HSUCCESSFUL RESULTS FROM F01MMSTRASSEN   )
      STOP

 2    WRITE(6,101)
 101  FORMAT(20X,17HINCORRECT RESULTS   )
      STOP

      END
```

DAP program

```
      ENTRY SUBROUTINE TESTMULT

      REAL U(,,2,2),V(,,2,2),W(,,2,2),X(,,2,2),RELDIFF(,,2,2)
      LOGICAL FLAG

      COMMON/TEST/A(,),B(,)
      COMMON/FLAG/FLAG

      EXTERNAL REAL MATRIX FUNCTION E
C
C     CALL CONVERSION ROUTINES
C
      CALL CONVFME(A)
      CALL CONVFME(B)
      FLAG = .TRUE.
C
C     GENERATE ENLARGED MATRIX DATA
C
      V(,,1,1) = A
      W(,,1,1) = B
      V(,,1,2) = V(,,1,1) * 3.1
      W(,,1,2) = W(,,1,1) + 6.3
      V(,,2,1) = W(,,1,1) * 0.9
      W(,,2,1) = V(,,1,1) * 2.4
      V(,,2,2) = V(,,1,2) + 5.6
      W(,,2,2) = W(,,1,2) * 1.3
C
C     CALL THE STRASSEN ROUTINE AND ANOTHER ROUTINE FOR
C     MATRIX MULTIPLICATION
C
      CALL F01_MM_STRASSEN(U,V,W)
      CALL MM2N(X,V,W)

C
C    CHECK THE TWO SETS OF RESULTS CALCULATED
C
      DO 11 L = 1,2
        DO 11 K = 1,2
          RELDIFF(,,K,L) = E(U(,,K,L),X(,,K,L))
          IF(ANY(RELDIFF(,,K,L).GT.0.0001))FLAG = .FALSE.
 11   CONTINUE
C
C     CONVERT DATA AND RETURN TO THE HOST
C
      CALL CONVSFL(FLAG,1)
      RETURN
      END
```

```
      REAL MATRIX FUNCTION E(X,Y)
C
C  FUNCTION TO COMPARE RELATIVE VALUES OF TWO MATRICES
C
      DIMENSION X(,),Y(,)

      E = X - Y
      X(ABS(X).LT.1.0E-50) = 1.0
      E(ABS(Y).GE.1.0E-50) = ABS(E/X)
      X(ABS(X - 1.0).LT.1.0E-50) = 0.0

      RETURN
      END

      SUBROUTINE MM2N(A,B,C)
C
C     THIS SUBROUTINE IS DESIGNED TO MULTIPLY TWO 64 X 64
C     MATRICES TOGETHER.THE METHOD USED TO PERFORM THIS TASK
C     IS THE "INTUITIVE" METHOD,THAT IS ,IMPLEMENTING THE
C     32 X 32 MATRIX MULTIPLICATION 8 TIMES TO COMPUTE EACH
C     PARTITION SEPARATELY.
C
      DIMENSION A(,,2,2),B(,,2,2),C(,,2,2)
      INTEGER K
C
C     INITIALISE THE RESULTANT ARRAY.
C
      A(,,1,1) = 0.0
      A(,,1,2) = 0.0
      A(,,2,1) = 0.0
      A(,,2,2) = 0.0
C
C     PERFORM THE MATRIX MULTIPLICATION FOR EACH PARTITION
C     IN TURN.
C
      DO 1 K = 1,32
        A(,,1,1)=A(,,1,1)+MATC(B(,K,1,1))*MATR(C(K,,1,1))
        A(,,1,1)=A(,,1,1)+MATC(B(,K,1,2))*MATR(C(K,,2,1))
        A(,,1,2)=A(,,1,2)+MATC(B(,K,1,1))*MATR(C(K,,1,2))
        A(,,1,2)=A(,,1,2)+MATC(B(,K,1,2))*MATR(C(K,,2,2))
        A(,,2,1)=A(,,2,1)+MATC(B(,K,2,1))*MATR(C(K,,1,1))
        A(,,2,1)=A(,,2,1)+MATC(B(,K,2,2))*MATR(C(K,,2,1))
        A(,,2,2)=A(,,2,2)+MATC(B(,K,2,1))*MATR(C(K,,1,2))
        A(,,2,2)=A(,,2,2)+MATC(B(,K,2,2))*MATR(C(K,,2,2))
 1    CONTINUE

      RETURN
      END
```

Results

```
                    SUCCESSFUL RESULTS FROM F01MMSTRASSEN
```

Chapter 6

F02 – Eigenvalues and eigenvectors

Contents:

6.1 F02_ALL_EIG_VALS_TD_ES

release 1

1 Purpose

F02_ALL_EIG_VALS_TD_ES uses Sturm sequences to find all the eigenvalues of a symmetric tridiagonal matrix of order up to 32.

2 Specification

```
SUBROUTINE F02_ALL_EIG_VALS_TD_ES (ALPHA , GAMMA , N , EVALS
    IC , IFAIL)
INTEGER N , IC , IFAIL
REAL ALPHA () , GAMMA () , EVALS()
```

3 Description

The algorithm uses the following theorem:

Given a symmetric tridiagonal matrix with diagonal elements $c_1, \ldots, c_n$ and off diagonal elements $b_2, \ldots, b_n$, then let the sequence $q_1(\lambda), \ldots, q_n(\lambda)$ be defined for any real λ by:

$$q_1(\lambda) = c_1 - \lambda$$

$$q_i(\lambda) = (c_i - \lambda) - \frac{b_i^2}{q_{i-1}(\lambda)} \qquad (i = 2, \ldots, n)$$

If $a(\lambda)$ is the number of negative $q_i(\lambda)$ then this number is equal to the number of eigenvalues less than λ. If $q_{i-1}(\lambda) = 0$ for any i, then it can be replaced in (4.2) by a suitably small non-zero value (see [1]). Also see [1] for an example of another use of this theorem.

For each eigenvalue, an initial interval is determined which is known to contain the eigenvalue. Each such interval is then repeatedly subdivided until further refinements produce no improvement in the corresponding eigenvalue or the subinterval width becomes less than 10^{-35}.

4 References

[1] BARTH W, MARTIN R S and WILKINSON J H

Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection: Numer Math 9, pp 386-393, 1967.

5 Arguments

ALPHA – REAL VECTOR

On entry ALPHA specifies the components of the main diagonal of the tridiagonal matrix, that is, ALPHA (I) = A (I, I) (I = 1, 2, ..., N). Elements (N + 1) to 32 may be undefined; the argument is unchanged on exit from the sub-routine.

GAMMA – REAL VECTOR

On entry GAMMA specifies the components of the off diagonal of the tridiagonal matrix, that is, GAMMA(I) = A(I, I + 1) = A(I + 1, I) (I = 2, 3, ..., N). Elements not in the range 2 to N may be undefined; the argument is unchanged on exit from the sub-routine.

N – INTEGER

On entry, N specifies the order of the tridiagonal matrix. N must lie in the range 2 to 32, and is unchanged on exit.

EVALS – REAL VECTOR

On exit, EVALS contains the N eigenvalues of the matrix in components 1 to N.

IC – INTEGER

On exit, IC contains the number of calls to the Sturm sequence evaluation routine required to isolate all the eigenvalues. Note: for each such call the Sturm sequence is evaluated at 1024 points simultaneously.

IFAIL – INTEGER

Unless the routine detects an error (see section 6) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N not in the range 2 to 32 inclusive
IFAIL = 2	After 10 calls to the Sturm sequence evaluation routine some eigenvalues have not converged

7 Auxiliary Routines

This routine calls the GS lbrary routines X02_EPSILON, X05_LONG_INDEX, X05_SHORT_INDEX and Z_F02_STURM_SEQ_1.

8 Accuracy

In general, you can expect at least 6 significant figures of accuracy in the computed eigenvalues.

9 Further Comments

None

10 Keywords

Eigenvalues, Sturm sequences, symmetric tridiagonal matrices

11 Example

The matrix used in the example is a tridiagonal matrix of the form:

```
a b
b a b
  b a b
    . . .
      . . .
        . . .
          . . .
```

the eigenvalues of which are given by:

$$\lambda_s = a+2b\cos\left(\frac{s\pi}{n+1}\right) \quad (s = 1,\, 2,\, \ldots, n)$$

The largest error in the computed solution is 6 parts in 10^7.

Host program

```
      PROGRAM MAINES
      REAL ALPHA(32), GAMMA(32), Y(32)
      COMMON /ALPHA/ALPHA /GAMMA/GAMMA /Y/Y
      COMMON/SCALARS/N,IC,IFAIL
      N = 32
      DO 10 I = 1,32
      ALPHA (I) = 5.0
10    GAMMA (I) = 10.0
      CALL DAPCON('entes.dd')
      CALL DAPSEN('SCALARS',N,1)
      CALL DAPSEN('ALPHA',ALPHA,32)
      CALL DAPSEN('GAMMA',GAMMA,32)
      CALL DAPENT('ENTES')
      CALL DAPREC('Y',Y,32)
      CALL DAPREC('SCALARS',N,3)
      CALL DAPREL
      WRITE(6,100) IFAIL,IC, (Y(I), I = 1,32)
100   FORMAT(' IFAIL =',I5/'  IC  =',I5/ ' EIGENVALUES'/(G14.7))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENTES
      REAL ALPHA(), GAMMA(), Y()
      COMMON /ALPHA/ALPHA /GAMMA/GAMMA /Y/Y
      COMMON /SCALARS/ N,IC,IFAIL
      CALL CONVFVE(ALPHA,32,1)
      CALL CONVFVE(GAMMA,32,1)
      CALL CONVFSI(N,1)
      CALL F02ALL_EIG_VALS_TD_ES(ALPHA,GAMMA,N,Y,IC,IFAIL)
      CALL CONVVFE(Y,32,1)
      CALL CONVSFI(N,3)
      RETURN
      END
```

Results

```
IFAIL = 0
IC    = 6
EIGENVALUES
-14.97665
-14.90632
-14.79012
```

6.2 F02_ALL_EIG_VALS_TD_LV

release 1

1 Purpose

F02_ALL_EIG_VALS_TD_LV uses Sturm sequences to find all the eigenvalues of a symmetric tridiagonal matrix of order up to 1024.

2 Specification

```
   SUBROUTINE F02_ALL_EIG_VALS_TD_LV(ALPHA , GAMMA , N , EVALS ,
+     IC , IFAIL)
   INTEGER N , IC , IFAIL
   REAL ALPHA(,) , GAMMA(,) , EVALS(,)
```

3 Description

The algorithm uses the following theorem:

Given a symmetric tridiagonal matrix with diagonal elements $c_1, \ldots, c_n$ and off diagonal elements $b_2, \ldots, b_n$, then let the sequence $q_1(\lambda), \ldots, q_n(\lambda)$ be defined for any real λ by:

$$q_1(\lambda) = c_1 - \lambda \tag{1}$$

$$q_i(\lambda) = (c_i - \lambda) - \frac{b_i^2}{q_{i-1}(\lambda)} \qquad (i = 2, \ldots, n) \tag{2}$$

If $a(\lambda)$ is the number of negative $q_i(\lambda)$ then this number is equal to the number of eigenvalues less than λ. If $q_{i-1}(\lambda) = 0$ for any i, then it can be replaced in (2) by a suitably small non-zero value (see [1]). Also see [1] for an example of another use of this theorem.

For each eigenvalue, an initial interval is determined which is known to contain the eigenvalue. Each such interval is then repeatedly subdivided until further refinements produce no improvement in the corresponding eigenvalue or the subinterval width becomes less than 10^{-35}.

4 References

[1] BARTH W, MARTIN R S and WILKINSON J H

Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection: Numer Math, 9, pp 386-393, 1967.

5 Arguments

ALPHA – REAL VECTOR

On entry ALPHA specifies the components of the main diagonal of the tridiagonal matrix, that is, ALPHA(I) = A(I, I) (I = 1, 2, ..., N). Elements (N + 1) to 1024 may be undefined; the argument is unchanged on exit from the sub-routine.

GAMMA – REAL VECTOR

On entry GAMMA specifies the components of the off diagonal of the tridiagonal matrix, that is, GAMMA(I) = A(I, I + 1) = A(I + 1, I) (I = 2, 3, ..., N). Elements not in the range 2 to N may be undefined; the argument is unchanged on exit from the sub-routine.

N – INTEGER

On entry, N specifies the order of the tridiagonal matrix. N must lie in the range 2 to 1024, and is unchanged on exit.

EVALS – REAL VECTOR

On exit, EVALS contains the N eigenvalues of the matrix in components 1 to N.

IC – INTEGER

On exit, IC contains the number of calls to the Sturm sequence evaluation routine required to isolate all the eigenvalues. Note: for each such call the Sturm sequence is evaluated at 1024 points simultaneously.

IFAIL – INTEGER

Unless the routine detects an error (see section 6) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N not in the range 2 to 1024 inclusive
IFAIL = 2	After 30 calls to ther Sturm sequence evaluation routine some eigenvalues have not converged

7 Auxiliary Routines

This routine calls the GS library routines X02_EPSILON, X05_LONG_INDEX, X05_SHORT_INDEX and Z_F02_STURM_SEQ_2.

8 Accuracy

In general, you can expect about 5 or 6 significant figures of accuracy in the computed eigenvalues.

9 Further Comments

None

10. Keywords

Eigenvalues, Sturm sequences, symmetric tridiagonal matrices

11 Example

The matrix used in the example is a tridiagonal matrix of the form:

```
a b
b a b
  b a b
    . . .
      . . .
        . . .
          . . .
```

the eigenvalues of which are given by:

$$\lambda_s = a+2b\cos\left(\frac{s\pi}{n+1}\right) \quad (s = 1, 2, \ldots, n)$$

Host program

```
      PROGRAM MAIN
      REAL ALPHA(1024),GAMMA(1024),EVALS(1024)
      COMMON  /MATS/ALPHA,GAMMA,EVALS
      COMMON  /SCALARS/N,IC,IFAIL
      N=128
      DO 10 I=1,128
      ALPHA(I)=5.0
10    GAMMA(I)=10.0
      CALL DAPCON('ent.dd')
      CALL DAPSEN('SCALARS',N,1)
      CALL DAPSEN('MATS',ALPHA,2*1024)
      CALL DAPENT('ENT')
      CALL DAPREC('MATS',ALPHA,3*1024)
      CALL DAPREC('SCALARS',N,3)
      CALL DAPREL
      WRITE(6,1000) IFAIL,IC,(EVALS(I),I=1,128)
1000  FORMAT('  IFAIL =',I5/' IC = ',I5/' EIGENVALUES'/(G14.7))
      STOP

      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL ALPHA(,),GAMMA(,),EVALS(,)
      COMMON  /MATS/ALPHA,GAMMA,EVALS
      COMMON  /SCALARS/N,IC,IFAIL
      CALL CONVFME (ALPHA)
      CALL CONVFME (GAMMA)
      CALL CONVFSI (N,1)
      CALL F02_ALL_EIG_VALS_TD_LV(ALPHA,GAMMA,N,EVALS,IC,IFAIL)
      CALL CONVMFE (EVALS)
      CALL CONVSFI (N,3)
      RETURN
      END
```

Results

```
IFAIL =    0
IC    =   20
EIGENVALUES
-14.99412
-14.97626
-14.94660
```

6.3 F02_EIG_VALS_TD_LV

release 1

1 Purpose

F02_EIG_VALS_TD_LV uses Sturm sequences to find up to 32 selected eigenvalues of a symmetric tridiagonal matrix of order up to 1024.

2 Specification

```
    SUBROUTINE F02_EIG_VALS_TD_LV(ALPHA , GAMMA , N , I_EIGS ,
 +      NUM_EIGS , EVALS , IC , IFAIL)
    INTEGER N , I_EIGS() , NUM_EIGS , IC , IFAIL
    REAL ALPHA(,) , GAMMA(,) , EVALS()
```

3 Description

The algorithm uses the following theorem:

Given a symmetric tridiagonal matrix with diagonal elements $c_1, \dots c_n$ and off diagonal elements $b_2, \dots b_n$, then let the sequence $q_1(\lambda), \dots q_n(\lambda)$ be defined for any real λ by:

$$q_1(\lambda) = c_1 - \lambda \qquad (1)$$

$$q_i(\lambda) = (c_i - \lambda) - \frac{b_i^2}{q_{i-1}(\lambda)} \quad (i = 2, \dots, n) \qquad (2)$$

If $a(\lambda)$ is the number of negative $q_i(\lambda)$ then this number is equal to the number of eigenvalues less than λ. If $q_{i-1}(\lambda) = 0$ for any i, then it can be replaced in (4.6) by a suitably small non-zero value (see [1]). Also see [1] for an example of another use of this theorem.

For each eigenvalue, an initial interval is determined which is known to contain the eigenvalue. Each such interval is then repeatedly subdivided until further refinements produce no improvement in the corresponding eigenvalue or the subinterval width becomes less than 10^{-35}.

4 References

[1] BARTH W, MARTIN R S and WILKINSON J H

Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. Numer. Math. 9 pp 386-393 (1967).

5 Arguments

ALPHA – REAL VECTOR

On entry ALPHA specifies the components of the main diagonal of the tridiagonal matrix, that is, ALPHA(I) = A(I, I) (I = 1, 2, ..., N). Elements (N + 1) to 1024 may be undefined; the argument is unchanged on exit from the sub-routine.

GAMMA – REAL VECTOR

On entry GAMMA specifies the components of the off diagonal of the tridiagonal matrix, that is, GAMMA(I) = A(I, I + 1) = A(I + 1, I) (I = 2, 3, ..., N). Elements not in the range 2 to N may be undefined; the argument is unchanged on exit from the sub-routine.

N – INTEGER

On entry, N specifies the order of the tridiagonal matrix. N must lie in the range 2 to 0124, and is unchanged on exit.

I_EIGS – INTEGER VECTOR

I_EIGS is used to indicate which eigenvalues of the matrix are required. If the eigenvalues are $l(1) <= l(2) <= \ldots <= l(N)$ then to determine the subset $l(j_1), l(j_2), \ldots, l(j_p)$ the first p (equals NUM_EIGS) components of I_EIGS must be set to $j_1, j_2, \ldots, j_p$ and the condition $j_1 < j_2 < \ldots < j_p$ must hold. Components $(p+1)$ to 32 may be undefined; the argument is unchanged on exit.

NUM_EIGS – INTEGER

On entry NUM_EIGS specifies the number of eigenvalues required and must be in the range 1 to 32; it is unchanged on exit.

EVALS – REAL VECTOR

On exit, EVALS contains the NUM_EIGS eigenvalues of the matrix in components 1 to NUM_EIGS.

IC – INTEGER

On exit, IC contains the number of calls to the Sturm sequence evaluation routine required to isolate all the eigenvalues. Note: for each such call the Sturm sequence is evaluated at 1024 points simultaneously.

IFAIL – INTEGER

Unless the routine detects an error (see section 6) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N not in the range 2 to 1024 inclusive
IFAIL = 2	Entries 1 to NUM_EIGS of I_EIGS are not strictly increasing or lie outside the range 1 to 1024
IFAIL = 3	After 10 calls to the Sturm sequence evaluation routine some eigenvalues have not converged

7 Auxiliary Routines

This routine calls the GS library routines X02_EPSILON, X05_LONG_INDEX, X05_SHORT_INDEX and Z_F02_STURM_SEQ_2.

8 Accuracy

In general, you can expect about 6 significant figures of accuracy in the computed eigenvalues.

9 Further Comments

None

10 Keywords

Eigenvalues, Sturm sequences, symmetric tridiagonal matrices

11 Example

The matrix used in the example is a tridiagonal matrix of the form:

```
a  b
b  a  b
   b  a  b
      .  .  .
         .  .  .
            .  .  .
               .  .  .
```

the eigenvalues of which are given by:

$$\lambda_s = a{+}2bcos\left(\frac{s\pi}{n+1}\right) \qquad (s = 1,\, 2,\, \ldots,\, n)$$

The eigenvalues requested are spread throughout the spectrum and the largest error in the computed solution was 7 parts 10^7.

Host program

```
      PROGRAM MAIN
      REAL ALPHA(1024), GAMMA(1024),Y(32)
      INTEGER IEIGS(32)
      COMMON /MATS/ALPHA,GAMMA
      COMMON /IEIGS/IEIGS /Y/Y
      COMMON /SCALS/N,NUMEIGS,IC,IFAIL
      N = 1024
      DO 10 I = 1,1024
      ALPHA(I) = 5.0
10    GAMMA (I) = 10.0
      NUMEIGS = 32
      DO 20 I = 1,32
20    IEIGS(I) = 32*I
      CALL DAPCON('ent.dd)
      CALL DAPSEN('MATS',ALPHA,2*1024)
      CALL DAPSEN('IEIGS',IEIGS,32)
      CALL DAPSEN('SCALS',N,2)
      CALL DAPENT('ENT')
      CALL DAPREC('SCALS',N,4)
      CALL DAPREC('Y',Y,32)
      CALL DAPREL
      WRITE(6,100) IFAIL,IC, (IEIGS(I),Y(I), I= 1,32)
100   FORMAT(' IFAIL =',I5/' IC =',I5/
     *'EIGENVALUES'/(I5,5X,G14.7))
      STOP
      END
```

DAP program

```
ENTRY SUBROUTINE ENT
INTEGER IEIGS()
REAL ALPHA(,), GAMMA(,), Y()
COMMON /MATS/ALPHA,GAMMA
COMMON /IEIGS/IEIGS  /Y/Y
COMMON /SCALS/N, NUMEIGS,IC,IFAIL
CALL CONVFME(ALPHA)
CALL CONVFME(GAMMA)
CALL CONVFVI(IEIGS,32,1)
CALL CONVFSI(N,2)
CALL F02_EIG_VALS_TD_LV(ALPHA,GAMMA,N,IEIGS,NUMEIGS,Y,IC,IFAIL)
CALL CONVVFE(Y,32,1)
CALL CONVSFI(N,4)
RETURN
END
```

Results

```
IFAIL = 0
IC = 6
EIGENVALUES
  32     -14.90388
  64     -14.61645
  96     -14.14048
 128     -13.48052
```

6.4 F02_JACOBI

release 1

1 Purpose

F02_JACOBI calculates the eigenvalues and eigenvectors of a real symmetric matrix of order 32 x 32.

The method is based on the classical Jacobi algorithm using plane rotations.

2 Specification

```
SUBROUTINE F02_JACOBI(C , EVALUES , Q , BOOL)
REAL C(,) , EVALUES() , Q(,)
LOGICAL BOOL
```

3 Description

The cyclic Jacobi method is a well known technique for determining the eigensolution of a matrix [4]. A real symmetric matrix A is reduced to diagonal form by application of plane rotations. Full details can be found in [2].

4 References

[1] MODI J J

Error analysis for the parallel Jacobi method: QMC internal report, Department of Computer Science and Statistics, Queen Mary College, Mile End Road, London, E1 4NS: available on request from the DAP Suppost Unit at Queen Mary College.

[2] MODI J J

Jacobi methods for eigenvalue and related problems in a parallel computing environment: Ph D thesis, University of London.

[3] SAMEH A H

On Jacobi and Jacobi-like algorithms for the parallel computer: Mathematics of Computation, v 25, no 115, pp 579-590, July 1971.

[4] WILKINSON J H

The Algebraic Eigenvalue Problem: Clarendon Press, Oxford, 1965.

5 Arguments

C – REAL MATRIX

On entry C contains the real symmetric matrix whose eigenvalues are required, and is unchanged on exit.

EVALUES – REAL VECTOR

On exit EVALUES will contain the eigenvalues of C, in ascending order.

Q – REAL MATRIX

If BOOL was set to .TRUE. on entry then on exit the columns of Q will contain the eigenvectors of C.

The eigenvector in column I corresponds to the I^{th} element of EVALUES.

BOOL – LOGICAL

If BOOL is set to .TRUE. on entry, the eigenvectors of C will be calculated as well as the eigenvalues; BOOL is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

This routine calls the GS lbrary routines M01_PERMUTE_COLS, M01_SORT_V_R4 and X05_PATTERN.

8 Accuracy

The method is numerically very stable (see [1]). Tests show that the routine agrees with EISPACK routines, run on a 60 bit word computer, to 4 or 5 significant figures.

9 Further Comments

None

10 Keywords

Disjoint Rotations, Jacobi Method, Parallel Algorithm.

11 Example

The example finds the eigensolution of a 32 x 32 matrix.

Host program

```
      PROGRAM MAINJACOBI
      LOGICAL BOOL
      COMMON /A/A(32,32) /EV/EIGENVALUES(32)
      COMMON /Q/Q(32,32) /BOOL/BOOL
      BOOL = .TRUE.
      DO 20 J = 1,32
      DO 20 I = 1,32
      A(I,J) = 0.0
      IF ((I + 1).EQ.J) A(I,J) = 1.0
      IF ((J + 1).EQ.I) A(I,J) = 1.0
 20   CONTINUE
      CALL DAPCON('v3.dd')
      CALL DAPSEN('A',a,1024)
      CALL DAPSEN('BOOL',BOOL,1)
      CALL DAPENT('V3')
      CALL DAPREC('EV',eigenvalues,32)
      WRITE (6,1000) (EIGENVALUES(I),I = 1,32)
```

```
1000    FORMAT (' Eigenvalues ' /(1X,F14.5))
        WRITE (6,1500)
1500    FORMAT(' Eigenvectors')
        CALL DAPREC('Q',Q,1024)
        CALL DAPREL
        J=1
        DO 40 I = 1,32
40      WRITE (6,2000) Q(I,J)
2000    FORMAT(1X,F14.5)
        STOP
        END
```

DAP program

```
        ENTRY SUBROUTINE V3
        REAL A(,) , Q(,) ,EIGENVALUES()
        LOGICAL BOOL
        COMMON /A/A /EV/EIGENVALUES
        COMMON /Q/Q /BOOL/BOOL
        CALL CONVFME(A)

        CALL CONVFSL(BOOL,1)
        CALL F02_JACOBI(A,EIGENVALUES,Q,BOOL)
        CALL CONVVFE(EIGENVALUES,32,1)
        CALL CONVMFE(Q)
        RETURN
        END
```

Results

```
 Eigenvalues
       -1.99084
       -1.96374
       -1.91889
       -1.85665
       -1.77758
       -1.68242
       -1.57204
       -1.44740
       -1.30967
       -1.16006
        -.99995
        -.83079
        -.65410
        -.47150
        -.28462
        -.09516
```

```
          .09516
          .28462
          .47150
          .65410
          .83079
          .99995
         1.16006
         1.30967
         1.44740
         1.57204
         1.68242
         1.77758
         1.85665
         1.91889
         1.96374
         1.99084
 Eigenvectors
         -.02315
          .04610
         -.06866
          .09063
         -.11182
          .13204
         -.15109
          .16879
         -.18499
          .19953
         -.21228
          .22313
         -.23198
          .23876
         -.24338
          .24582
         -.24603
          .24401
         -.23977
          .23334
         -.22477
          .21414
         -.20154
          .18710
         -.17093
          .15319
         -.13404
          .11365
         -.09221
          .06992
         -.04697
          .02360
```

Chapter 7

F04 – Simultaneous linear equations

Contents:

7.1 F04_BIGSOLVE

release 1

1 Purpose

F04_BIGSOLVE is a routine for solving large sets of linear equations. The maximum size of the system depends on the size of the DAP store – for a 32 by 32 DAP with a 4 Mbyte store this maximum size is 1023, whereas for a 32 by 32 DAP with an 8 Mbyte store the maximum size is 1407. The method used was developed by D Hunt; it consists of a block form of Gauss Elimination with column pivoting. The matrix of the coefficients of the equations is of size 'SIZE' by 'SIZE' and the right hand side is assumed to be held in column 'SIZE' + 1. The whole matrix is held in the DAP partitioned in DAPSIZE blocks.

You are not recommended to use this routine for systems of order 32 or less – for which you should use the routine F04_GJ_NLE_ES.

2 Specification

```
SUBROUTINE F04_BIGSOLVE(BIGM , SIZE , ALLBLKS , IFAIL)
REAL BIGM(,,ALLBLKS,ALLBLKS)
INTEGER SIZE , IFAIL , ALLBLKS
```

3 Description

You can use this routine to solve a system of equations of maximum size N = 1023 on the 4 Mbyte 32 by 32 DAP, (N = 1407 on the 8 Mbyte 32 by 32 DAP) using a block form of Gauss elimination with column pivoting [2]. After the forward step, the matrix is conceptually of the form: (illustrated for a hypothetical 4 by 4 DAP and for N = 11)

```
1 0 0 0   X X X X   X X X X
0 1 0 0   X X X X   X X X X
0 0 1 0   X X X X   X X X X
0 0 0 1   X X X X   X X X X

0 0 0 0   1 0 0 0   X X X X
0 0 0 0   0 1 0 0   X X X X
0 0 0 0   0 0 1 0   X X X X
0 0 0 0   0 0 0 1   X X X X

0 0 0 0   0 0 0 0   1 0 0 X
0 0 0 0   0 0 0 0   0 1 0 X
0 0 0 0   0 0 0 0   0 0 1 X
0 0 0 0   0 0 0 0   0 0 0 0

  ( X = non zero value)
```

Gauss Jordan elimination is used for the diagonal blocks (see [1]). In practice, the diagonal and below diagonal blocks are not needed and are therefore left undefined.

On DAP the relevant part of the pivot column will in general be spread over several sheets. In DAP 500 that part of the pivot column is extracted in order to find the maximum in a single operation.

The factors by which the rows of the large matrix are multiplied are obtained by dividing the pivot column by the pivot element. This is done in a single matrix division operation on the extracted data.

The solution time is ultimately $O(m^3 \text{ x } d)$, where the matrix is partitioned into m by m sheets each of size d by d to match the DAP 500 array. (In terms of the parameters below, $N =$SIZE, $((m-1)\,d < N < md)$ and $m =$ALLBLKS).

4 References

[1] FOX, L

Numerical Linear Algebra: Chapters 3, 7, Oxford University Press, Oxford,1964

[2] HUNT, D J

Solution of a large system of equations on DAP using a hybrid Gauss/Gauss Jordan method: DAPSU Technical Report 7.27: available on request from The DAP Support Unit, Queen Mary Collge, Mile End Road, London E1 4NS

[3] PARKINSON, D and LIDDELL, H M

The measurement of performance on a highly parallel system: IEEE Trans on Computers, Special Issue, Nov 1982

5 Arguments

BIGM – REAL MATRIX array of dimension (, ,ALLBLKS,ALLBLKS)

On entry the first SIZE rows and columns must be set to the elements of the matrix of coefficients of the equations defining the linear system. The right-handside of the equations is stored in column SIZE + 1. The values in BIGM are changed during execution of the subroutine, and on exit column SIZE + 1 contains the solution of the system.

SIZE – INTEGER

On entry SIZE must be set to the order of the system. Unchanged on exit. SIZE must not be less than 2.

ALLBLKS – INTEGER

On entry ALLBLKS must be set to the number of DAP partitions needed to store the complete system (i.e. including the RHS). Unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1 SIZE is less than 2

IFAIL = 2 One of the conditions:

32*(ALLBLKS - 1) < SIZE
32*ALLBLKS - 1 >= SIZE

has been violated

6 Error Indicators – continued

IFAIL = 3 A zero pivot has been found during the back substitution process. The calculation is terminated

IFAIL = 4 A very small pivot has been found during the back substitution process and the matrix is probably singular.

Computation proceeds anyway, but the results should be treated with caution

7 Auxiliary Routines

None

8 Accuracy

The accuracy depends on the conditioning of the system; during extensive testing of this single precision implementation of the routine the maximum residual was approximately 10^{-3}.

9 Further Comments

None

10 Keywords

Gauss elimination, Gauss-Jordan, linear solver.

11 Example

Host program

```
      PROGRAM HOSTBIGSOLVER
      COMMON/INPUT1/A(32,32,5,5)
      COMMON/STATS/FNMONE,FNMTWO,FNMINF
      COMMON/IFAIL/IFAIL

      DATA N,IX/32,1111111/

      CALL DAPCON('bigtest.dd')
      CALL INITDATA(N,IX)
      CALL DAPSEN('INPUT1',A,25*1024)
      CALL DAPENT('BIGSOLVETEST')
      CALL DAPREC('IFAIL',IFAIL,1)
      CALL DAPREC('STATS',FNMONE,3)
      CALL DAPREL
      WRITE(6,99)IFAIL
 99   FORMAT(10X,7HIFAIL =,I3)
      IF(IFAIL.EQ.1.OR.IFAIL.EQ.2.OR.IFAIL.EQ.3)STOP
      WRITE(6,100)FNMONE,FNMTWO,FNMINF
```

```
 100  FORMAT(20H SUM OF RESIDUALS = ,E10.4//
     131H SUM OF SQUARES OF RESIDUALS = ,E10.4//
     220H MAXIMUM RESIDUAL = ,E10.4)
      STOP
      END
```

DAP program

```
      SUBROUTINE INITDATA(N,IX)
      COMMON/INPUT1/A(32,32,5,5)
C
C     THIS SUBROUTINE CREATES THE INITIAL SEEDS THAT THE DAP CAN USE TO
C     CALCULATE EXACTLY THE REQUIRED SET OF PSEUDO-RANDOM NUMBERS.
C     THIS IS DONE IN ORDER TO BE ABLE TO MAKE FAIR COMPARISONS IN
C     RESPECT OF RUNTIME AS WELL AS NUMERICAL RESULTS
C
      DO 1 L = 1,5
      DO 1 K = 1,5
      DO 1 J = 1,N
      DO 1 I = 1,N
        IY =FLOAT(IX)/22369.624
        IX=125*IX-2796203*IY
        A(I,J,K,L) = FLOAT(IX)/2796203.
 1    CONTINUE
      RETURN
      END

      ENTRY SUBROUTINE BIGSOLVETEST
      COMMON/INPUT1/A(,,5,5)
      COMMON/STATS/FNMONE,FNMTWO,FNMINF
      COMMON/IFAIL/IFAIL

      REAL BIGM(,,5,5),QSAVE(,5),TRHS(,5),RESIDU(,5),MAXIMUM(,5)
      REAL MULT(,),X(,5)
      INTEGER N, IFAIL, DAPSIZE,RHSCOL

      NDAPS = 5
      DO 700 L = 1,NDAPS
      DO 700 K = 1,NDAPS
        CALL CONVFME (A( , ,K,L))
 700  CONTINUE
```

```
      DAPSIZE = 32
      N = 150
      RHSCOL = N - (NDAPS - 1)*DAPSIZE + 1
      DO 400 L = 1,NDAPS
      DO 400 K = 1,NDAPS
        BIGM( , ,K,L) = A( , ,K,L)

400   CONTINUE
      DO 500 L = 1,NDAPS
      QSAVE( ,L) = A( ,RHSCOL,L,NDAPS)
500   CONTINUE

      CALL F04_BIGSOLVE(BIGM,N,NDAPS,IFAIL)
      IF(IFAIL.EQ.0.OR.IFAIL.EQ.4)GO TO 200
      CALL CONVSFI(IFAIL,1)
      RETURN

200   CONTINUE
      DO 300 K = 1,NDAPS
      X( ,K) = BIGM( ,RHSCOL,K,NDAPS)
300   CONTINUE

      FNMONE = 0.
      FNMTWO = 0.
      FNMINF = 0.
      DO 60 K = 1,NDAPS
      TRHS( ,K) = 0.
      DO 70 L = 1,NDAPS
        MULT = MATR( X( ,L))
        TRHS( ,K)  = TRHS( ,K) + SUMC(MULT*A(,,K,L))
70      CONTINUE
      RESIDU( ,K) = ABS(TRHS( ,K) - QSAVE( ,K))
      IF(K .NE. NDAPS) GO TO 80
      DO 90 I = RHSCOL,DAPSIZE
        RESIDU(I,NDAPS) = 0.0
        QSAVE(I,NDAPS) = 0.0
        TRHS(I,NDAPS) = 0.0
90     CONTINUE
80     CONTINUE
      FNMONE = FNMONE + SUM(RESIDU( ,K))
      FNMTWO = FNMTWO + SUM( RESIDU( ,K)**2)
      MAXIMUM( ,K) = 0.
      MAXIMUM(RESIDU( ,K) .GT.MAXIMUM( ,K) ,K) = RESIDU( ,K)
      IF (MAXV(MAXIMUM( ,K)).GT.FNMINF) FNMINF = MAXV(MAXIMUM( ,K))
60    CONTINUE
600   CONTINUE
      CALL CONVSFE(FNMONE,3)
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Results

```
         IFAIL = 0
SUM OF RESIDUALS = 0.9086E-01
SUM OF SQUARES OF RESIDUALS = 0.7045E-06
MAXIMUM RESIDUAL = 0.1943E-03
```

7.2 F04_GJ_NLE_ES

release 1

1 Purpose

F04_GJ_NLE_ES is a routine for solving the system of linear equations $Ax = b$ for x, where A is a non sparse matrix of order N in the range 1 to 32, using the Gauss Jordan method. It is not particularly efficient for small values of N.

2 Specification

```
SUBROUTINE F04_GJ_NLE_ES(A , X , Q , N , IFAIL)
REAL A(,) , X() , Q()
INTEGER N , IFAIL
```

3 Description

The Gauss Jordan method [1,2] can be considered as a variant of Gauss elimination, but the elimination is also applied to terms above the diagonal at each stage.

For example, for a 4 by 4 system:

```
Step 0          X X X X  =  X
                X X X X  =  X
                X X X X  =  X
                X X X X  =  X

Step 1          X X X X  =  X
                0 X X X  =  X
                0 X X X  =  X
                0 X X X  =  X
```

(This is the same as in Gauss elimination)

```
Step 2          X 0 X X  =  X
                0 X X X  =  X
                0 0 X X  =  X
                0 0 X X  =  X

Step 3          X 0 0 X  =  X
                0 X 0 X  =  X
                0 0 X X  =  X
                0 0 X X  =  X

Step 4          X 0 0 0  =  X
                0 X 0 0  =  X
                0 0 X 0  =  X
                0 0 0 X  =  X
```

(X represents a non zero value)

Thus the parallelism at each step is maximised and there is no need to perform the back substitution. On a computer with $m\,x\,m$ parallel processors, where m exceeds the number of equations, N, the operation count for Gauss Jordan is N divisions, multiplications and subtractions, which is the same number of operations required by the elimination phase of Gauss elimination. However, the latter also requires $N-1$ multiplies and subtractions for the back substitution phase. On a serial machine, the operation count for Gauss Jordan is $O\,(\frac{N^3}{2})$, which is greater than that for Gauss elimination – $O\,(\frac{N^3}{3})$. The back substitution phase takes $O\,(N^2)$ operations and is therefore negligible for large systems.

4 References

[1] FLANDERS P M , HUNT D J, REDDAWAY S F and PARKINSON D

Efficient high speed computing with the distributed array processor, in High Speed Computer and Algorithm Organisation: Academic Press, London, 1977

[2] WEBB S J

Solution of elliptic partial differential equations on the ICL Distributed Array Processor: ICL Technical Journal, vol 2, 175 – 189 (1980)

5 Arguments

A – REAL MATRIX

On entry, elements $A_{(i,j)}$ $(i, j = 1, \ldots, N)$ must be set to the elements of the matrix defining the linear system. The argument is unchanged on exit.

X – REAL VECTOR

On exit the first N elements of X will contain the solution of the system.

Q – REAL VECTOR

On entry, the first N elements of Q should contain the values of the right hand side (b) of the system. The argument is unchanged on exit.

N – INTEGER

On entry, N must be set to the order of the system; it is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N is not in the range 1 to 32.
IFAIL = 2	A zero pivot has been found. The calculation is terminated.
IFAIL = 3	A very small pivot has been found and the matrix is probably singular. Computation proceeds anyway, but the results should be treated with caution.

7 Auxiliary Routines

None

8 Accuracy

Accuracy depends on the conditioning of the system; during testing of this single precision implementation, the maximum residual was less than 10^{-3}.

9 Further Comments

None

10 Keywords

Gauss Jordan, linear system solver

11 Example

Host program

```
      PROGRAM HOSTSOLVER
      COMMON/INPUTD1/A(32,32)
      COMMON/INPUTD2/Q(32),X(32)
      COMMON/STATS/FNMONE,FNMTWO,FNMINF
      COMMON /IFAIL/IFAIL

      DATA N,IX/32,1111111/

      CALL INITDATA(N,IX)
      CALL DAPCON('gjtest.dd')
      CALL DAPSEN('INPUTDATA1',a,1024)
      CALL DAPSEN('INPUTDATA2',Q,32)
      CALL DAPENT('GJTEST')
      CALL DAPREC('IFAIL',IFAIL,1)
      CALL DAPSEN('INPUTDATA2',x,32)
      WRITE(6,200)IFAIL
 200  FORMAT(10X,8H IFAIL =,I2)
      IF(IFAIL.NE.0)STOP
      CALL DAPREC('STATS',FNMONE,3)
      CALL DAPREL
      WRITE(6,100)FNMONE,FNMTWO,FNMINF
 100  FORMAT(20H SUM OF RESIDUALS = ,E10.4//
     131H SUM OF SQUARES OF RESIDUALS = ,E10.4//
     220H MAXIMUM RESIDUAL = ,E10.4)
      STOP
      END

      SUBROUTINE INITDATA(N,IX)
      COMMON/INPUTD1/A(32,32)
      COMMON/INPUTD2/Q(32),X(32)
```

```
C
C     THIS SUBROUTINE CREATES THE INITIAL SEEDS THAT THE DAP CAN USE
C     TO CALCULATE EXACTLY THE REQUIRED SET OF PSEUDO-RANDOM NUMBERS.
C     THIS IS DONE IN ORDER TO BE ABLE TO MAKE FAIR COMPARISONS IN
C     RESPECT OF RUNTIME AS WELL AS NUMERICAL RESULTS
C
      DO 1 I = 1,N
      DO 1 J = 1,N
        IY =FLOAT(IX)/22369.624
        IX=125*IX-2796203*IY
        A(I,J) = FLOAT(IX)/2796203
 1    CONTINUE
      DO 2 I = 1,N
        IY = FLOAT(IX)/22369.624
        IX=125*IX-2796203*IY
        Q(I) = FLOAT(IX)/2796203
 2    CONTINUE
      RETURN
      END
```

DAP program

```
      ENTRY SUBROUTINE GJTEST
      COMMON/INPUTDATA1/A(,)
      COMMON/INPUTDATA2/Q(),X()
      COMMON/STATS/FNMONE,FNMTWO,FNMINF
      COMMON/IFAIL/IFAIL

      REAL ASAVE(,),QSAVE(),TRHS(),RESIDU(),MAXIMUM(),MULT(,)
     + ,QSAVE1()
      LOGICAL MASK(,),VMASK()

      CALL CONVFME(A)
      CALL CONVFVE(Q,32,1)
      ASAVE = A
      QSAVE = Q
      QSAVE1 = QSAVE

      N = 27
      MASK = ROWS(1,N).AND.COLS(1,N)
      VMASK = ELS(1,N)
      QSAVE = QSAVE1
      Q(VMASK) = QSAVE
      Q(.NOT.VMASK) = 0.
      A(MASK) = ASAVE
      A(.NOT.MASK) = 0.
```

```
      CALL F04_GJ_NLE_ES(A,X,Q,N,IFAIL)
      X(.NOT.VMASK) = 0.
      QSAVE(.NOT.VMASK) = 0.
      IF(IFAIL.NE.0)GO TO 100
      TRHS =0.
      MULT=MATR(X)
      TRHS = SUMC(MULT*ASAVE)
      TRHS(.NOT.VMASK) = 0.
      RESIDU = ABS(TRHS - QSAVE)
      FNMONE=SUM(RESIDU)
      FNMTWO= SUM(RESIDU**2)
      MAXIMUM = 0.
      MAXIMUM(RESIDU.GT.MAXIMUM) = RESIDU
      FNMINF = MAXV(MAXIMUM)

      CALL CONVVFE(X,32,1)
      CALL CONVSFE(FNMONE,3)
 100  CONTINUE
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Results

```
        IFAIL = 0
SUM OF RESIDUALS = 0.3069

SUM OF SQUARES OF RESIDUALS = 0.3604E-06

MAXIMUM RESIDUAL = 0.1466E-03
```

7.3 F04_QR_GIVENS_SOLVE

release 1

1 Purpose

F04_QR_GIVENS_SOLVE solves the linear system $Ax = b$ for x, where A is an n by n matrix with $2 < n < 33$. The routine may be used to solve simultaneously for up to 32 different right hand side vectors b.

2 Specification

```
SUBROUTINE F04_QR_GIVENS_SOLVE(A , X , B , N , NB , IFAIL)
INTEGER N , NB , IFAIL
REAL A(,) , X(,) , B(,)
```

3 Description

The routine factorizes the given n by n matrix A as:

$$QA = R$$

where Q is an orthogonal matrix and r is upper triangular.

Givens method of plane rotations is used to annihilate elements of A below the leading diagonal until the matrix R remains. This leaves an upper triangular system which is solved by back substitution. Row i of A is used to annihilate the element in position $(i+1, j)$ by pre-multiplying A by a matrix of the form:

$$p^{j}_{(i,i+1)} = diag\,(I_{(i-1)}, U_{(i,i+1)}, I_{(n-i-1)}) \qquad 1 \leq j \leq n-1$$

where $U_{(i,i+1)} = \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}$, with $c_i^2 + s_i^2 = 1$

In the usual serial application, these rotations are applied sequentially, but on the DAP you can perform up to $\frac{n}{2}$ rotations simultaneously [1].

4 References

[1] SAMEH A H and KUCK D J

On stable parallel linear system solvers: Journal of the Association of Computing Machinery, vol 25, no 1, pp 81-91.

5 Arguments

A – REAL MATRIX

On entry, elements $A_{(i,j)}$ $(i = 1, 2, \ldots, N; j = 1, 2, \ldots, N)$ must be set to the elements of the matrix defining the linear system. A is unchanged on exit.

X – REAL MATRIX

On exit, column i of X will contain the solution of the system corresponding to the i^{th} column of B.

5 Arguments – continued

B – REAL MATRIX

On entry, columns 1 to NB must give the NB right hand side vectors. B is unchanged on exit.

N – INTEGER

On entry, N must be set to the order of the matrix A. N is unchanged on exit.

NB – INTEGER

On entry, NB must be set to the number of right hand side vectors for which the system is to be solved. NB is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	N is not in the range 3 to 32 or NB is not in the range 1 to 32
IFAIL = 2	A zero pivot has been found during the back substitution process, that is, the matrix is singular
IFAIL = 3	A very small pivot has been found during the back substitution process and the matrix is probably singular. Computation proceeds anyway, but you should treat the results with caution

7 Auxiliary Routines

This routine calls the DAP library routines Z_F04_BACK_SUBST, Z_FO4_SPREAD_LMAT_EAST, Z_FO4_SPREAD_RMAT_EAST and Z_FO4_UPDATE.

8 Accuracy

Empirical results indicate that errors may be expected in the 6^{th} or 7^{th} significant digit. The routine will return IFAIL = 3 (see **Error Indicators** above) if the condition:

$$\frac{MAX_{i,j}\ |R_{ij}|}{MIN_i\ |R_{ii}|} > 5 \times 10^5$$

is satisfied, where R_{ij} is the upper triangular matrix defined in **Description** above.

9 Further Comments

You must not use common blocks with the names:

C_F04_QR1 and C_F04_QR2

10 Keywords

Givens' rotation, linear equations

11 Example

The example solves a 5 by 5 linear system with one right hand side. The true solution vector is $[1,1,1,1,1]^T$.

Host program

```
      PROGRAM MAINGIVEN
      REAL A(32,32),X(32,32),B(32,32)
      COMMON /MATS/A,X,B
      COMMON /SCALARS/ N,NB,IFAIL
      READ(5,*) N,NB
      READ(5,*) ((A(I,J),J=1,N), I=1,N)
      READ(5,*) ((B(I,J),J=1,NB),I=1,N)
      CALL DAPCON('entgiven.dd')
      CALL DAPSEN('SCALARS',N,3)
      CALL DAPSEN('MATS',A,3*1024)
      CALL DAPENT('ENTGIVEN')
      CALL DAPREC('SCALARS',N,3)
      CALL DAPREC('MATS',A,2*1024)
      CALL DAPREL
      WRITE (6,1000) IFAIL
1000  FORMAT( ' IFAIL = ',I5)
      IF (IFAIL.NE.0 .AND. IFAIL.NE.3) STOP
      WRITE(6,2000) ((X(I,J),J=1,NB),I=1,N)
2000  FORMAT(/' Solution:'/(1X,F12.7))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENTGIVEN
      REAL A(,),X(,),B(,)
      COMMON /MATS/A,X,B
      COMMON /SCALARS/N,NB,IFAIL
      CALL CONVFME(A)
      CALL CONVFME(B)
      CALL CONVFSI(N,3)
      CALL F04_QR_GIVENS_SOLVE(A,X,B,N,NB,IFAIL)
      CALL CONVMFE(X)
      CALL CONVSFI(N,3)
      RETURN
      END
```

Data

```
 5  1

 3.0  -7.0  1.5  2.5  6.1
 8.0   1.6  0.0 -3.0  2.8
-0.5   1.6  2.3  7.4 -8.5
 0.0  -1.0 -2.3  1.7  5.8
 2.7   1.3 -3.5  0.0  4.1

 6.1   9.4  2.3  4.2  4.6
```

Results

```
IFAIL = 0

Solution:
   0.9999998
   0.9999985
   0.9999961
   0.9999998
   0.9999990
```

7.4 F04_TRIDS_ES

release 1

1 Purpose

F04_TRIDS_ES returns the solution of a tridiagonal linear system of equations of order up to 32. That is, it finds vector x where:

$$Mx = y$$

and M is a tridiagonal matrix.

2 Specification

REAL VECTOR FUNCTION F04_TRIDS_ES (A , B , C , Y , N , IFAIL)
INTEGER N , IFAIL
REAL A ( ) , B ( ) , C ( ) , Y ( )

3 Description

The algorithm used is of the recursive doubling type. At each step the distance of the outer diagonals from the main diagonal is doubled. When only a diagonal matrix remains the solution is obtained by a simple division. Full details may be found in [1].

4 References

[1] WHITEWAY J

A parallel algorithm for solving tridiagonal systems: DAPSU Newsletter, 3 December 1979: available on request from the DAP Support Unit, Queen Mary College, Mile End Road, London E1 4NS

5 Arguments

A – REAL VECTOR

On entry, elements 2 to N of A must be set to the values of the lower diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $A(I)$ must be set to $M(I,I-1)$ $(I = 2, \dots, N)$. Elements with subscripts not in the range 2 to N are ignored. A is unchanged on exit.

B – REAL VECTOR

On entry, elements 1 to N of B must be set to the values of the main diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $B(I)$ must be set to $M(I,I)$ $(I = 1, \dots, N)$. Elements with subscripts not in the range 1 to N are ignored. B is unchanged on exit.

C – REAL VECTOR

On entry, elements 1 to $N-1$ of C must be set to the values of the upper diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $C(I)$ must be set to $M(I,I+1)$ $(I = 1, \dots, N-1)$. Elements with subscripts not in the range 1 to $N-1$ are ignored. C is unchanged on exit.

Y – REAL VECTOR

On entry, elements 1 to N of Y must be set to the values of the RHS vector. Elements with subscripts not in the range 1 to N are ignored. Y is unchanged on exit.

5 Arguments – continued

N – INTEGER

On entry, N must specify the size of the system (in the range 2 to 32). That is, for $Mx = y$, M must be N by N.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1 At some stage during the calculation, an element on the leading diagonal is zero. This implies the original matrix was singular. The contents of F04_TRIDS_ES in this case are undefined

IFAIL = 2 At some stage during the calculation, the matrix has ceased to be diagonally dominant. Note: this is only a warning and the routine continues to completion (if possible)

IFAIL = 3 N is not in the range 2 to 32

7 Auxiliary Routines

None

8 Accuracy

General results seem to indicate that the more diagonally dominant the system is the more accurate the results. IFAIL = 1 is possible for non-diagonally dominant systems even if the system is non-singular.

9 Further Comments

None

10 Keywords

Tridiagonal linear systems

11 Example

The example given is such that the solution vector should be 1. The system is diagonally dominant.

Host program

```
PROGRAM MAINTRIDSES
REAL ANS(32)
COMMON /ANS/ANS/IFAIL/IFAIL
CALL DAPCON('tridses.dd')
CALL DAPENT('ENTTRIDSES')
CALL DAPREC('ANS',ANS,32)
CALL DAPREC('IFAIL',IFAIL,1)
CALL DAPREL
WRITE (6,1000) IFAIL
```

```
1000   FORMAT(' IFAIL =',I5)
           IF (IFAIL.NE.0) STOP
           WRITE(6,2000) (ANS(I), I=1,15)
2000   FORMAT(' RESULTS'//(F12.7))
           STOP
           END
```

DAP program

```
      ENTRY SUBROUTINE ENTTRIDSES
      REAL LOWER(), UPPER(), DIAG(), ANS(), RHS()
      COMMON /ANS/ANS/IFAIL/IFAIL
      EXTERNAL REAL VECTOR FUNCTION F04_TRIDS_ES
      N = 15
      LOWER = 0.5
      UPPER = 0.5
      DIAG = 2.0
      RHS = 3.0
      RHS(1) = 2.5
      RHS(N) = 2.5
      ANS = F04_TRIDS_ES(LOWER,DIAG,UPPER,RHS,N,IFAIL)
      CALL CONVVFE(ANS,32,1)
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Results

```
   IFAIL =    0
 RESULTS

    .9999999
    .9999999
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
   1.0000000
    .9999999
    .9999999
```

7.5 F04_TRIDS_ES_SQ

release 1

1 Purpose

F04_TRIDS_ES_SQ returns the solution of a set of up to 32 tridiagonal linear systems of equations each of order up to 32. That is, it solves up to 32 systems of the form:

$$Mx = y$$

where M is a tridiagonal matrix.

2 Specification

```
REAL MATRIX FUNCTION F04_TRIDS_ES_SQ (A, B, C, Y, N, K, IFAIL)
INTEGER N , K , IFAIL
REAL A (,) , B (,) , C (,) , Y (,)
```

3 Description

The algorithm used is of the recursive doubling type. At each step the distance of the two outer diagonals from the main diagonal is doubled. When only a diagonal matrix remains the solution is obtained by a simple division. Each system is stored down the columns of the matrix arguments and so, many systems can be solved simultaneously. Full details can be found in [1].

4 References

[1] WHITEWAY J

A parallel algorithm for solving tridiagonal systems: DAPSU Newletter 3, December 1979: available from the DAP Support Unit, Queen Mary College, Mile End Road, London E1 4NS.

5 Arguments

A – REAL MATRIX

On entry, elements 2 to N of columns 1 to K of A must be set to the values of the lower diagonal of each of the K systems That is, if the K^{th} matrix is $M = m(i,j)$ then $A(I,K)$ must be set to $M(I, I-1)$ $(I = 2, 3, \ldots, N)$. Elements with row subscripts not in the range 2 to N or columns subscripts not in the range 1 to K are ignored. A is unchanged on exit.

B – REAL MATRIX

On entry, elements 1 to N of columns 1 to K of B must be set to the values of the main diagonal of each of the K systems. That is, if the K^{th} matrix is $M = m(i,j)$ then $B(I,K)$ must be set to $M(I,I)(I = 1, 2, \ldots, N)$. Elements with row subscripts not in the range 1 to N or column subscripts not in the range 1 to K are ignored. B is unchanged on exit.

C – REAL MATRIX

On entry, elements 1 to $N-1$ of columns 1 to K of C must be set to the values of the upper diagonal of each of the K systems. That is, if the K^{th} matrix is $M = m(i,j)$ then $C(I,K)$ must be set to $M(I, I+1)$ $(I = 1, 2, \ldots, N-1)$. Elements with row subscripts not in the range 1 to $N-1$ or column subscripts not in the range 1 to K are ignored. C is unchanged on exit.

Y – REAL MATRIX

On entry, elements 1 to N of columns 1 to K of Y must be set to the values of the K RHS vectors. Elements with row subscripts not in the range 1 to N or column subscripts not in the range 1 to K are ignored. Y is unchanged on exit.

N – INTEGER

On entry, N must specify the order of the tridiagonal systems (in the range 1 to 32).

K – INTEGER

On entry, K must specify the number of tridiagonal systems to be solved (in the range 1 to 32).

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	At some stage during the calculation, an element on one of the leading diagonals is zero. This implies that, at least, one of the systems was singular. The contents of F04_TRIDS_ES_SQ in this case are undefined
IFAIL = 2	As a minimum, at some stage during the calculation, one matrix has ceased to be diagonally dominant. Note : this is only a warning and the routine continues to completion (if possible)
IFAIL = 3	N is not in the range 1 to 32 or K is not in the range 1 to 32

7 Auxiliary Routines

None

8 Accuracy

General results seem to indicate that the more diagonally dominant the systems are the more accurate the results. IFAIL = 1 is possible for non-diagonally dominant systems even if the system is non-singular.

9 Further Comments

None

10 Keywords

Tridiagonal linear systems

11 Example

The example given solves 2 tridiagonal systems of order 15. The solutions are 1 and 2 respectively.

Host program

```
      PROGRAM MAINTRIDSESSQ
      REAL ANS(32,32)
      COMMON /ANS/ANS/IFAIL/IFAIL
      CALL DAPCON('tridsessq.dd')
      CALL DAPENT('ENTTRIDSESSQ')
      CALL DAPREC('ANS',ANS,1024)
      CALL DAPREC('IFAIL',IFAIL,1)
      CALL DAPREL
      WRITE(6,1000) IFAIL
1000  FORMAT (' IFAIL =',I5)
      IF (IFAIL.NE.0.AND.IFAIL.NE.2) STOP
      WRITE(6,2000)  (ANS(I,1), ANS(I,2), I = 1,15)
2000  FORMAT(' RESULTS'//(2F12.7))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENTTRIDSESSQ
      REAL LOWER(,), UPPER(,), DIAG(,), RHS(,), ANS(,)
      COMMON /ANS/ANS/IFAIL/IFAIL
      EXTERNAL REAL MATRIX FUNCTION F04_TRIDS_ES_SQ
      N = 15
      K = 2
      LOWER = 0.5
      UPPER = 0.5
      DIAG = 2.0
      RHS(,1) = 3.0
      RHS(,2) = 6.0
      RHS(1,1) = 2.5
      RHS(N,1) = 2.5
      RHS(1,2) = 5.0
      RHS(N,2) = 5.0
      ANS = F04_TRIDS_ES_SQ (LOWER,DIAG,UPPER,RHS,N,K,IFAIL)
      CALL CONVMFE(ANS)
      CALL CONVFSI(IFAIL,1)
      RETURN
      END
```

Results

```
IFAIL =    0
RESULTS

  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000048
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
  1.0000019   2.0000019
```

7.6 F04_TRIDS_LV

release 1

1 Purpose

F04_TRIDS_LV returns the solution of a tridiagonal linear system of equations of order up to 1024. That is, it finds vector x where:

$$Mx = y$$

and M is a tridiagonal matrix.

2 Specification

```
REAL MATRIX FUNCTION F04_TRIDS_LV(A , B , C , Y , N , IFAIL)
INTEGER N , IFAIL
REAL A(,) , B(,) , C(,) , Y(,)
```

3 Description

The algorithm used is of the recursive doubling type. At each step the distance of the two outer diagonals from the main diagonal is doubled. When only a diagonal matrix remains the solution is obtained by a simple division. Full details may be found in [1].

4 References

[1] WHITEWAY J

A parallel algorithm for solving tridiagonal systems: DAPSU Newsletter 3, December 1979: available from the DAP Support Unit, Queen Mary College, Mile End Road, London E1 4NS.

5 Arguments

A – REAL MATRIX

On entry, elements 2 to N of A (treated as a long vector) must be set to the values of the lower diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $A(I)$ must be set to $M(I, I-1)$ $(I = 2, 3, \ldots, N)$. Elements with subscripts not in the range 2 to N are ignored. A is unchanged on exit.

B — REAL MATRIX

On entry, elements 1 to N of B (treated as a long vector) must be set to the values of the main diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $B(I)$ must be set to $M(I,I)$ $(I = 1, 2, \ldots, N)$. Elements with subscripts not in the range 1 to N are ignored. B is unchanged on exit.

C – REAL MATRIX

On entry, elements 1 to $N-1$ of C (treated as a long vector) must be set to the values of the upper diagonal of the tridiagonal matrix. That is, if the matrix is $M = m(i,j)$ then $C(I)$ must be set to $M(I, I+1)$ $(I = 1, 2, \ldots, N-1)$. Elements with subscripts not in the range 1 to $N-1$ are ignored. C is unchanged on exit.

Y – REAL MATRIX

On entry, elements 1 to N of Y (treated as a long vector) must be set to the values of the RHS vector. Elements with subscripts not in the range 1 to N are ignored. Y is unchanged on exit.

N – INTEGER

On entry, N must specify the size of the system (in the range 2 to 1024). That is, for $Mx = y$, M must be N by N.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	At some stage during the calculation, an element on the leading diagonal is zero. This implies the original matrix was singular. The contents of F04_TRIDS_LV in this case are undefined
IFAIL = 2	At some stage during the calculation, the matrix has ceased to be diagonally dominant. Note: this is only a warning and the routine continues to completion (if possible)
IFAIL = 3	N is not in the range 2 to 1024

7 Auxiliary Routines

None

8 Accuracy

General results seem to indicate that the more diagonally dominant the system is the more accurate the results. IFAIL = 1 is possible for non-diagonally dominant systems even if the system is non-singular.

9 Further Comments

None

10 Keywords

Tridiagonal linear systems

11 Example

The example given is such that the solution vector should be 1. The system is diagonally dominant.

Host program

```
PROGRAM MAINTRIDS_LV
REAL ANS(1024)
COMMON/ANS/ANS/IFAIL/IFAIL
CALL DAPCON('tridslv.dd')
CALL DAPENT('ENTTRIDS_LV')
CALL DAPREC('ANS',ANS,1024)
CALL DAPREC('IFAIL',IFAIL,1)
CALL DAPREL
```

```
      WRITE (6,1000) IFAIL
1000  FORMAT (' IFAIL =',I5)
      IF (IFAIL.NE.0) STOP
      WRITE(6,2000) (ANS(I), I = 1, 15)
2000  FORMAT(' RESULTS'// (F12.7))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENTTRIDS_LV
      REAL LOWER(,), UPPER(,), DIAG(,), ANS(,), RHS(,)
      COMMON /ANS/ANS/IFAIL/IFAIL
      EXTERNAL REAL MATRIX FUNCTION F04_TRIDS_LV
      N = 15
      LOWER = 0.5
      UPPER = 0.5
      DIAG = 2.0
      RHS = 3.0
      RHS(1) = 2.5
      RHS(N) = 2.5
      ANS = F04_TRIDS_LV(LOWER,DIAG,UPPER,RHS,N,IFAIL)
      CALL CONVMFE(ANS)
      CALL CONVSFI(IFAIL,1)
      RETURN
      END
```

Results

```
            IFAIL =    0

      RESULTS
        1.00000020
        1.00000020
        1.00000020
          .
          .
          .
```

All other results are also equal to 1.0000020

Chapter 8

G05 – Random numbers

Contents:

8.1 G05_MC_BEGIN

release 1

1 Purpose

G05_MC_BEGIN sets the basic generator routine G05_MC_I8 to an initial state.

2 Specification

SUBROUTINE G05_MC_BEGIN

3 Description

This routine sets the internal variable N used by G05_MC_I8 to the value 123456789 x $(2^{32} + 1)$.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244 (1985)

5 Arguments

None

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

The routine uses a labelled COMMON block C_G05_MC.

10 Keywords

Initialisation, random numbers

11 Example

The example program prints the first five pseudo-random real numbers from a uniform distribution between 0 and 1, generated by G05_MC_R4 after initialization by G05_MC_BEGIN.

Host program

```
PROGRAM MAIN

REAL*4 RAND(1024)
COMMON/RESULT/RAND
```

```
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,1024)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT('G05_MC_BEGIN EXAMPLE PROGRAM RESULTS'/1X/
     *5(1X,F10.4/))

      STOP
      END
```

DAP Program

```
      ENTRY SUBROUTINE ENT

      REAL*4 RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL REAL*4 MATRIX FUNCTION G05_MC_R4

      CALL G05_MC_BEGIN
      RAND=G05_MC_R4(0.0)
      CALL CONVMFE(RAND)

      RETURN
      END
```

Results

```
 G05_MC_BEGIN EXAMPLE PROGRAM RESULTS

     0.6149
     0.8745
     0.1511
     0.0734
     0.2451
```

8.2 G05_MC_I4

release 1

1 Purpose

G05_MC_I4 returns an INTEGER*4 MATRIX containing 1024 pseudo-random integer numbers taken from a uniform distribution between 0 and $2^{31} - 1$.

2 Specification

```
INTEGER*4 MATRIX FUNCTION G05_MC_I4 (I)
INTEGER*4 I
```

3 Description

The routine calls G05_MC_I8 which uses the multiplicative congruential method:

$$N = 13^{13}\ N \bmod 2^{59}$$
$$\text{G05\_MC\_I4} = N/2^{28}$$

where N is a variable, internal to G05_MC_I8, whose value is preserved between calls of the routine. Its initial value is set by a call to either G05_MC_BEGIN or G05_MC_REPEAT.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244, 1985

5 Arguments

I - INTEGER*4

A dummy argument required by FORTRAN-PLUS syntax

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routine G05_MC_I8.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Pseudo-random number, random number, rectangular distribution, uniform distribution

11 Example

The example program prints the first five pseudo-random numbers from a uniform distribution between 0 and $2^{31}-1$, generated by G05_MC_I4 after initialization by G05_MC_BEGIN.

Host Program

```
      PROGRAM MAIN

      INTEGER*4 RAND(1024)
      COMMON/RESULT/RAND

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,1024)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT(/' G05_MC_I4 EXAMPLE PROGRAM RESULTS'/1X/
     *       5(1X,I20/))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      INTEGER*4 RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL INTEGER*4 MATRIX FUNCTION G05_MC_I4

      CALL G05_MC_BEGIN
      RAND=G05_MC_I4(0)
      CALL CONVMFI(RAND)

      RETURN
      END
```

Results

```
 G05_MC_I4 EXAMPLE PROGRAM RESULTS

           1815152335
            436969313
            976973459
           1028379600
           1443266400
```

8.3 G05_MC_I8

release 1

1 Purpose

G05_MC_I8 returns an INTEGER*8 MATRIX containing 1024 pseudo-random integer numbers taken from a uniform distribution between 10 and $2^{59} - 1$.

2 Specification

```
INTEGER*8 MATRIX FUNCTION G05_MC_I8 (I)
INTEGER*8 I
```

3 Description

The routine uses the multiplicative congruential method:

$$N = 13^{13} \text{ N mod } 2^{59}$$
$$\text{G05\_MC\_I8} = N$$

where N is a variable, internal to G05_MC_I8, whose value is preserved between calls of the routine. Its initial value is set by a call to either G05_MC_BEGIN or G05_MC_REPEAT.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244, 1985

5 Arguments

I – INTEGER*8

A dummy argument required by FORTRAN-PLUS syntax

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

The routine uses labelled COMMON block C_G05_MC.

10 Keywords

Pseudo-random number, random number, rectangular distribution, uniform distribution

11 Example

This FORTRAN-PLUS fragment traces the pseudo-random numbers from a uniform distribution between 0 and $2^{59} - 1$ generated by G05_MC_I8 after initialization by G05_MC_BEGIN.

DAP program

```
ENTRY SUBROUTINE ENT

INTEGER*8 RAND(,)

EXTERNAL INTEGER*8 MATRIX FUNCTION G05_MC_I8

CALL G05_MC_BEGIN
RAND=G05_MC_I8(0)
TRACE 1(RAND)

RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 9

Integer Matrix Local Variable RAND in 64 bits - addressed by Stack + 0.09

(Row 01 Col 01)   487251244993469717,   476067912847080853,
       (Col 03)   190484975398149653,   493464185425411733,
       (Col 05)   517514364922158869,   463547216227221397,
          .
          .
```

There are 512 lines of detailed output altogether.

8.4 G05_MC_NORMAL_R4

release 1

1 Purpose

G05_MC_NORMAL_R4 provides a REAL*4 matrix containing normal pseudo-random variates from the distribution N(0,1).

2 Specification

```
REAL*4 MATRIX FUNCTION G05_MC_NORMAL_R4(D)
REAL*4 D
```

3 Description

The real matrix G05_MC_NORMAL_R4 is set equal to 1024 of either of:

$$\mathrm{SQRT}(-2.0\ \mathrm{LOG}(U_1))\ \mathrm{SIN}(2\pi\ U_2)$$

$$\mathrm{SQRT}(-2.0\ \mathrm{LOG}(U_1))\ \mathrm{COS}(2\pi\ U_2)$$

where U_1 and U_2 are uniform pseudo-random numbers generated by G05_MC_R4 (see Atkinson[1]).

4 References

[1] ATKINSON A C and PEARCE M C

The computer generation of Beta, Gamma and Normal random variables: J R Statist Soc 139, pp 431-461, 1976

5 Arguments

D – REAL*4

D is a dummy argument required by FORTRAN-PLUS syntax.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routine G05_MC_R4.

8 Accuracy

Not applicable

9 Further Comments

The routine uses the labelled COMMON block C_G05_N_NORM.

10 Keywords

Gaussian distribution, normal distribution, random numbers

11 Example

This example program prints the first five pseudo-random normal variates from a normal distribution with mean 0 and standard deviation 1, generated by G05_MC_NORMAL_R4 after initialization by G05_MC_BEGIN.

Host program

```
      PROGRAM MAIN

      REAL*4 RAND(1024)
      COMMON/RESULT/RAND

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,1024)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT(/,' G05_MC_NORMAL_R4 EXAMPLE PROGRAM RESULTS'/1X/
     *5(1X,F10.4/))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      REAL*4 RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL REAL*4 MATRIX FUNCTION G05_MC_NORMAL_R4

      CALL G05_MC_BEGIN
      RAND=G05_MC_NORMAL_R4(0.0)
      CALL CONVMFE(RAND)

      RETURN
      END
```

Results

```
 G05_MC_NORMAL_R4 EXAMPLE PROGRAM RESULTS

    -1.4384
     1.7104
      .1361
      .1528
     -.8427
```

8.5 G05_MC_R4

release 1

1 Purpose

G05_MC_R4 returns a REAL*4 MATRIX of 1024 pseudo-random real numbers taken from a uniform distribution between 0 and 1.

2 Specification

```
REAL*4 MATRIX FUNCTION G05_MC_R4 (X)
REAL*4 X
```

3 Description

The routine returns the matrix of values:

$$N/2^{59}$$

where N is the result of a call to G05_MC_I8.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244, 1985

5 Arguments

X – REAL*4

A dummy argument required by FORTRAN-PLUS syntax

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routine G05_MC_R8.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Pseudo-random number, random number, rectangular distribution, uniform distribution

11 Example

The example program prints the first five pseudo-random real numbers from a uniform distribution between 0 and 1, generated by G05_MC_R4 after initialization by G05_MC_BEGIN.

Host program

```
      PROGRAM MAIN

      REAL*4 RAND(1024)
      COMMON/RESULT/RAND

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,1024)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT(/,' G05_MC_R4 EXAMPLE PROGRAM RESULTS'/1X/
     *5(1X,F10.4/))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      REAL*4 RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL REAL*4 MATRIX FUNCTION G05_MC_R4

      CALL G05_MC_BEGIN
      RAND=G05_MC_R4(0.0)
      CALL CONVMFE(RAND)

      RETURN
      END
```

Results

```
 G05_MC_R4 EXAMPLE PROGRAM RESULTS

      .8452
      .2035
      .4549
      .4789
      .6721
```

8.6 G05_MC_R8

release 1

1 Purpose

G05_MC_R8 returns a REAL*8 MATRIX of 1024 pseudo-random real numbers taken from a uniform distribution between 0 and 1.

2 Specification

```
REAL*8 MATRIX FUNCTION G05_MC_R8(X)
REAL*8 X
```

3 Description

The routine returns the matrix of values:

$$N/2^{59}$$

where N is the result of a call to G05_MC_I8.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244, 1985

5 Arguments

X – REAL*8

A dummy argument required by FORTRAN-PLUS syntax

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routine G05_MC_I8.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Pseudo-random number, random number, rectangular distribution, uniform distribution

11 Example

The example program prints the first five pseudo-random real numbers from a uniform distribution between 0 and 1, generated by G05_MC_R8 after initialization by G05_MC_BEGIN.

Host program

```
      PROGRAM MAIN

      DOUBLE PRECISION RAND(1024)
      COMMON/RESULT/RAND

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,2048)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT(/,' G05_MC_R8 EXAMPLE PROGRAM RESULTS'/1X/
     *5(1X,F10.4/))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      DOUBLE PRECISION  RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL REAL*8 MATRIX FUNCTION G05_MC_R8

      CALL G05_MC_BEGIN
      RAND=G05_MC_R8(0.0)
      CALL CONVMFD(RAND)

      RETURN
      END
```

Results

```
 G05_MC_R8 EXAMPLE PROGRAM RESULTS

      .8452
      .2035
      .4549
      .4789
      .6721
```

8.7 G05_MC_REPEAT

release 1

1 Purpose

G05_MC_REPEAT sets the basic generator routine G05_MC_I8 to a repeatable initial state.

2 Specification

```
SUBROUTINE G05_MC_REPEAT( I )
INTEGER*4 I
```

3 Description

The routine sets the internal variable N used by G05_MC_I8 to a value calculated from the parameter I, where:

$$N = 2\,\mathrm{ABS}(I) + 1$$

The routine will yield different subsequent sequences of random numbers if called with different values of I, but the sequences will be repeatable in different runs of the calling program.

4 References

[1] SMITH K A, REDDAWAY S F and SCOTT D M

Very High Performance Pseudo-Random Number Generator on DAP: Computer Physics Communications, vol 37, pp 239-244, 1985

5 Arguments

I – INTEGER*4

On entry I specifies a number from which the new internal generator is calculated; I is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

The routine uses a labelled COMMON block C_G05_MC.

10 Keywords

Pseudo-random number, random number, rectangular distribution, uniform distribution

11 Example

The example program prints the first five pseudo-random real numbers from a uniform distribution between 0 and 1, generated by G05_MC_R4 after initialization by G05_MC_REPEAT.

Host program

```
      PROGRAM MAIN

      REAL*4 RAND(1024)
      COMMON/RESULT/RAND

      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('RESULT',RAND,1024)
      CALL DAPREL

      WRITE(6,1000)(RAND(I),I=1,5)
1000  FORMAT(/,' G05_MC_REPEAT EXAMPLE PROGRAM RESULTS'/1X/
     *5(1X,F10.4/))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      REAL*4 RAND(,)
      COMMON/RESULT/RAND

      EXTERNAL REAL*4 MATRIX FUNCTION G05_MC_R4

      CALL G05_MC_REPEAT(10)
      RAND=G05_MC_R4(0.0)
      CALL CONVMFE(RAND)

      RETURN
      END
```

Results

```
 G05_MC_REPEAT EXAMPLE PROGRAM RESULTS

      .6178
      .6430
      .5399
      .3852
      .1947
```

Chapter 9

H01 – Operations research, graph structures, networks

Contents:

9.1 H01_L_ASSIGN

release 1

1 Purpose

H01_L_ASSIGN solves the linear assignment problem with a minimum objective function and a real cost matrix of order N x N, where N $\leq$ 32.

2 Specification

```
SUBROUTINE H01_L_ASSIGN(C , X , N , MIN , IFAIL)
REAL C(,) , MIN
INTEGER X() , N , IFAIL
```

3 Description

The algorithm used is that of Ford and Fulkerson, [1], [2], which uses the Primal-Dual method. After dualizing the Primal problem, the routine aims to find a pair X, (U,V) of Primal and Dual solutions respectively which satisfy the complimentary slackness condition.

To find the appropriate solutions, a network G(U, V) is set up. There is an arc (i, j) in the graph whenever $u_i + v_j = c_{ij}$, where c_{ij} is the cost of assigning i to j. Next, the labelling algorithm of Ford and Fulkerson is appplied to find a maximum flow in G(U, V). If the maximum flow saturates the sink or (source), the problem is solved, otherwise the dual solutions are updated and the process restarts.

4 References

[1] DANTZIG G B

Linear Programming and Extensions: Princeton University Press, 1963

[2] FORD L R and FULKERSON D R

Flows in Networks: Princeton University Press, 1962

5 Arguments

C – REAL MATRIX

On entry C contains the N x N assignment cost matrix; C is unchanged on exit.

X – INTEGER VECTOR

On exit, X specifies the assignment solution; that is, if X(I) = J, for I, J $\leq$ N, then I is assigned to J.

N – INTEGER

On entry N is the order of the cost matrix C. N must lie between 2 and 32, and is unchanged on exit.

MIN – REAL

On exit MIN contains the assignment value.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1   N does not lie in the range 2 to 32

7 Auxiliary Routines

The routine calls the GS library routines X05_E_MIN_VC and X05_E_MIN_VR.

8 Accuracy

You can expect the computed value of the objective function MIN to be accurate to about 6 significant digits.

9 Further Comments

None

10 Keywords

Labelling algorithm, linear assignment, maximum flow, Primal-Dual algorithms

11 Example

The example is a 5 x 5 assignment problem, where the cost matrix is as follows:

$$C = \begin{vmatrix} 3 & 2 & 3 & 4 & 1 \\ 4 & 1 & 2 & 4 & 2 \\ 1 & 0 & 5 & 3 & 2 \\ 7 & 5 & 0 & 1 & 3 \\ 0 & 4 & 1 & 2 & 3 \end{vmatrix}$$

Hence N = 5

Host program

```
      PROGRAM LASP

      REAL C(32,32),MIN
      INTEGER X(32),N,IFAIL
      COMMON/A1/C
      COMMON/A2/X
      COMMON/A3/N,IFAIL
      COMMON/A4/MIN

      READ(*,*) N
      DO 10 I=1,N
   10 READ(*,*) (C(I,J), J=1,N)

      CALL DAPCON('initial.dd')
      CALL DAPSEN('A1',C,1024)
      CALL DAPSEN('A3',N,1)
```

```
      CALL DAPENT('INITIAL')

      CALL DAPREC('A1',C,1024)
      CALL DAPREC('A2',X,32)
      CALL DAPREC('A3',N,2)
      CALL DAPREC('A4',MIN,1)

      CALL DAPREL

      WRITE (*,*) 'IFAIL = ',IFAIL

      IF (IFAIL .NE. 0) STOP

      WRITE(6,30) MIN, (X(I), I=1,N)

   30 FORMAT(/,' MINIMUM VALUE OF ASP. =',F12.5,//,' THE ASSIGNMENTS',
     *          ' ARE AS FOLLOWS:',//, (1X,16I4))

      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE INITIAL

      REAL C(,),MIN
      INTEGER X( ),N,IFAIL
      COMMON/A1/C
      COMMON/A2/X
      COMMON/A3/N,IFAIL
      COMMON/A4/MIN

      CALL CONVFSI(N,1)
      CALL CONVFME(C)

      CALL H01_L_ASSIGN(C,X,N,MIN,IFAIL)

      CALL CONVMFE(C)
      CALL CONVVFI(X,32,1)
      CALL CONVSFI(N,2)
      CALL CONVSFE(MIN,1)

      RETURN
      END
```

Data

```
5
3  2  3  4  1
4  1  2  4  2
1  0  5  3  2
7  5  0  1  3
0  4  1  2  3
```

Results

```
IFAIL =  0

MINIMUM VALUE OF ASP. =     4.00000

THE ASSIGNMENTS ARE AS FOLLOWS:

   5    3    2    4    1
```

Chapter 10

J06 – Plotting

Contents:

10.1 J06_CHAR_CONT

release 1

1 Purpose

J06_CHAR_CONT returns a character matrix containing a rough contour map of a real matrix. You can control the number of contours and contour levels.

2 Specification

```
   SUBROUTINE J06_CHAR_CONT(A , MAP , CODE , LEVELS , NUM_LEVELS ,
+     IFAIL)
   INTEGER NUM_LEVELS , IFAIL
   REAL A (,) , LEVELS ( )
   CHARACTER MAP (,) , CODE ( )
```

3 Description

The routine adds contours one by one in order of descending height. For each contour the routine finds the area of the map which is less than the contour height. The border of this area is then found by eliminating any elements lying entirely within the area. This border is then taken as the contour.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, A contains the matrix for which a contour map is required. A is unchanged on exit.

MAP – CHARACTER MATRIX

On exit, MAP contains the required contour map.

CODE – CHARACTER VECTOR

On entry, CODE must either have been set to all spaces or the first NUM_LEVELS entries must contain the characters required to represent the contour levels. If CODE is all spaces then the default character sequence of 0123456789ABCDEFGHIJKLMNOPQRSTU-VWXYZ will be used. CODE is unchanged on exit.

LEVELS – REAL VECTOR

On entry, LEVELS must contain the NUM_LEVELS contour height values required (if NUM_LEVELS is positive), or may be undefined if NUM_LEVELS is negative.

If NUM_LEVELS is positive, successive entries in LEVELS must be strictly increasing.

On exit, elements 1 to ABS(NUM_LEVELS) of LEVELS contain the contour height values used in the contour plot, (other elements of LEVELS are undefined).

NUM_LEVELS – INTEGER

On entry, NUM_LEVELS specifies the number of contour lines required. NUM_LEVELS must not be zero or greater than 36 in absolute magnitude.

If NUM_LEVELS is positive, the contour heights will be taken from the vector LEVELS. If NUM_LEVELS is negative, ABS(NUM_LEVELS) contours will be drawn equally spaced between the maximum and minimum values of A. NUM_LEVELS is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	NUM_LEVELS is zero or not in the ranges -36 to -1 or 1 to 36
IFAIL = 2	The first NUM_LEVELS entries of LEVELS are not in strictly ascending order
IFAIL = 3	NUM_LEVELS is negative and all the entries in A are identical

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Contour plots

11 Example

The example generates two maps of the function $x^2 + y^2$, the first using the default character set and equally spaced contour heights and the second using heights and characters you define. The maps are output using the FORTRAN-PLUS TRACE statement.

Host program

```
PROGRAM MAIN
CALL DAPCON('example.dd')
CALL DAPENT('EXAMPLE')
CALL DAPREL
STOP
END
```

DAP program

```
ENTRY SUBROUTINE EXAMPLE
REAL A(,),CLEVELS()
INTEGER IV()
CHARACTER MAP(,),MYCODE()
CALL X05_SHORT_INDEX(IV,0)
A=FLOAT(MATR(IV-32)**2 + MATC(IV-32)**2)
CALL J06_CHAR_CONT(A,MAP,VEC(' '),CLEVELS,-10,IFAIL)
TRACE 1 (MAP,IFAIL,CLEVELS)
CLEVELS(1)=100.0
```

```
CLEVELS(2)=500.0
CLEVELS(3)=1000.0
CLEVELS(4)=1200.0
MYCODE(1)='A'
MYCODE(2)='B'
MYCODE(3)='C'
MYCODE(4)='D'
CALL J06_CHAR_CONT(A,MAP,MYCODE,CLEVELS,4,IFAIL)
TRACE 1 (MAP,IFAIL,CLEVELS)
RETURN
END
```

10.2 J06_ZEBRA_CHART release 1

1 Purpose

J06_ZEBRA_CHART returns a contour map suitable for output to a printing device of a real matrix. The output is called a ZEBRA chart; it consists of alternating bands of blanks and a given character.

2 Specification

```
CHARACTER MATRIX FUNCTION J06_ZEBRA_CHART(X , STEPS , CHAR)
INTEGER STEPS
REAL X (,)
CHARACTER CHAR
```

3 Description

The method used is straightforward: the input variable is scaled and divided into STEPS levels, and the least significant bit of the level number is used as a mask to create the output.

4 References

None

5 Arguments

X – REAL MATRIX

On entry, X contains the matrix to be plotted, and is unchanged on exit.

STEPS – INTEGER

On entry, STEPS specifies the number of bands in the chart (between the minimum and maximum of X), and is unchanged on exit.

CHAR – CHARACTER

On entry, CHAR specifies the character to be used in the bands, and is unchanged on exit.

6 Error Indicators

Errors detected by the routine:

If STEPS is less than 2 or the range of X is less than 1.0E-5 then a chart of all 'E's is produced.

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Contour map, Zebra chart

11 Example

The example calculates a simple function and uses the FORTRAN-PLUS TRACE facility to output the Zebra chart generated.

Host program

```
PROGRAM MAIN
CALL DAPCON('example.dd')
CALL DAPENT('EXAMPLE')
CALL DAPREL
STOP
END
```

DAP program

```
ENTRY SUBROUTINE EXAMPLE
EXTERNAL CHARACTER MATRIX FUNCTION J06_ZEBRA_CHART
REAL X(,)
CHARACTER OUT(,)
INTEGER I()
F=3.14159/32.
G=2.0*F
CALL SHORT_INDEX(I)
X=MATR(SIN(F*I))+MATC(COS(G*I))
OUT=J06_ZEBRA_CHART(X,10,'*')
TRACE 1 (OUT)
RETURN
END
```

Chapter 11

M01 – Sorting

Contents:

11.1 M01_BSORT_LV

release 1

1 Purpose

M01_BSORT_LV is a sorting routine based on bitonic sorting. Data is sorted according to a key, or the key alone may be sorted.

2 Specification

```
SUBROUTINE M01_BSORT_LV(KEY , L , X , D)
INTEGER KEY(,) , L , D
LOGICAL X(,,D)
```

3 Description

The routine uses Batcher's bitonic sorting algorithm. For a description see [1].

4 References

[1] KNUTH D E

The Art of Computer Programming, Vol 3 (Sorting and Searching): p 232 Addison-Wesley, 1973

5 Arguments

KEY – INTEGER MATRIX

On entry, KEY (considered as a long vector) must be defined as the key to the sort; on exit the contents of KEY will have been sorted.

L – INTEGER

On entry, L must have been set to zero if only the KEY is to be sorted; any other value will cause the data to be sorted as well. L is unchanged on exit.

X – <*any type*> MATRIX (or MATRIX array)

On entry, X contains the data to be sorted. On exit, X contains the sorted data.

D – INTEGER

On entry, D specifies the number of bit planes in the data, and is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Batcher sort, bitonic sort, data sort, key sort

11 Example

The example sorts 6 real values according to an integer key. Key entries beyond the data of interest are set to a large number to prevent them being involved in the sort.

Host program

```
      PROGRAM MAIN

      REAL DATA(1024)
      INTEGER KEY(1024)
      COMMON /KEY/KEY   /DATA/DATA

      DO 10 J=1,1024
   10 KEY(J)=10000
      READ(*,*) (KEY(I),I=1,6)
      READ(*,*) (DATA(I),I=1,6)
      WRITE(6,1000) (DATA(I),I=1,6),(KEY(I),I=1,6)
 1000 FORMAT(' INPUT VALUES:'//' DATA:',6F10.3/'  KEY:',6I10)

      CALL DAPCON('ent.dd')
      CALL DAPSEN('KEY',KEY,1024)
      CALL DAPSEN('DATA',DATA,1024)

      CALL DAPENT('ENT')

      CALL DAPREC('KEY',KEY,1024)
      CALL DAPREC('DATA',DATA,1024)

      CALL DAPREL

      WRITE(6,2000) (DATA(I),I=1,6),(KEY(I),I=1,6)
 2000 FORMAT(//' OUTPUT VALUES:'//' DATA:',6F10.3/'  KEY:',6I10)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      INTEGER KEY(,)
      REAL DATA(,)
      COMMON /KEY/KEY  /DATA/DATA
```

```
CALL CONVFMI(KEY)
CALL CONVFME(DATA)

CALL M01_BSORT_LV(KEY,1,DATA,32)

CALL CONVMFI(KEY)
CALL CONVMFE(DATA)

RETURN
END
```

Data

```
   8   -1    7    16    2   -3
 7.5   22  -81    -2    3   19
```

Results

```
INPUT VALUES:

DATA:     7.500    22.000   -81.000    -2.000     3.000    19.000
 KEY:         8        -1         7        16         2        -3

OUTPUT VALUES:

DATA:    19.000    22.000     3.000   -81.000     7.500    -2.000
 KEY:        -3        -1         2         7         8        16
```

11.2 M01_INV_PERMUTE_COLS

release 1

1 Purpose

M01_INV_PERMUTE_COLS permutes the first M columns of a matrix according to a permutation vector (IV). The result is equivalent to the FORTRAN-PLUS statements:

```
      DO 10 I = 1, M
   10 A_PERMUTED (,IV (I)) = A (,I)
```

2 Specification

```
SUBROUTINE M01_INV_PERMUTE_COLS (A , AP , IV , N , M)
INTEGER IV ( ) , N , M
<any type> A (,) , AP (,)
```

3 Description

Columns are permuted according to the integer index vector IV, such that column I is moved to column IV(I).

4 References

None

5 Arguments

A – <*any type*> MATRIX

On entry, A contains the matrix whose columns are to be permuted. A may be of any type, and is unchanged on exit.

AP – <*any type*> MATRIX

On exit, AP contains the columns of A permuted according to the index vector IV. AP should usually be of the same type as A. If M is less than 32, columns M+1 to 32 are unchanged on exit.

IV – INTEGER VECTOR

On entry, IV contains the required permutation, that is, column I of A will be moved to column IV(I) of AP. Elements 1 to M of IV must be in the range 1 to 32. If the entries of IV are not all distinct – for example, if IV(I) = IV(J) with J > I – then column AP(,IV(J)) will have the value A(,J) on exit. IV is unchanged on exit.

N – INTEGER

On entry, N contains the number of planes in the matrix to be permuted; possible values for N are:

N = 1	for permuting a logical matrix
N = 8	for permuting a character matrix
N = 8**n*	for permuting an INTEGER**n* or REAL**n* matrix

N should be less than 257, and is unchanged on exit.

5 Arguments – continued

M – INTEGER

On entry, M must contain a value in the range 1 to 32; only the first M index values of IV are used. M is unchanged on exit..

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_M01_AUX.

8 Accuracy

Not applicable

9 Further Comments

The parameters given as A and AP may be single arrays or part of a matrix set. For example, in:

CALL M01_INV_PERMUTE_COLS (L (,,5), LL(,,10), IV,1,32)

L and LL are logical matrix sets of size (at least) 5 and 10 respectively.

You must not use a common block with the names of CZ_M01_HEX1F or CZ_M01_REVERSE.

10 Keywords

Permutation

11 Example

The following FORTRAN-PLUS fragment reverses the order of the columns of a real matrix, that is,
AP = REVR(A).

```
      ENTRY SUBROUTINE ENT
      REAL A(,), AP(,)
      INTEGER IV()
      DO 10 I=1, 32
   10 IV(I) = 33 -- I
      DO 20 I = 1, 32
      DO 20 J = 1, 32
   20 A(I,J) = FLOAT (I + J)
      CALL M01_INV_PERMUTE_COLS (A, AP, IV, 32, 32)
      TRACE 1 (AP)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 10

Real Matrix Local Variable AP in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)  3.3000000E+01,  3.2000000E+01,  3.1000000E+01,
       (Col 04)  3.0000000E+01,  2.9000000E+01,  2.8000000E+01,
       (Col 07)  2.7000000E+01,  2.6000000E+01,  2.5000000E+01,
       (Col 10)  2.4000000E+01,  2.3000000E+01,  2.2000000E+01,
       (Col 13)  2.1000000E+01,  2.0000000E+01,  1.9000000E+01,
       (Col 16)  1.8000000E+01,  1.7000000E+01,  1.6000000E+01,
       (Col 19)  1.5000000E+01,  1.4000000E+01,  1.3000000E+01,
       (Col 22)  1.2000000E+01,  1.1000000E+01,  1.0000000E+01,
       (Col 25)  9.0000000E+00,  8.0000000E+00,  7.0000000E+00,
       (Col 28)  6.0000000E+00,  5.0000000E+00,  4.0000000E+00,
       (Col 31)  3.0000000E+00,  2.0000000E+00
(Row 02 Col 01)  3.4000000E+01,  3.3000000E+01,  3.2000000E+01,
       (Col 04)  3.1000000E+01,  3.0000000E+01,  2.9000000E+01,
       (Col 07)  2.8000000E+01,  2.7000000E+01,  2.6000000E+01,
       (Col 10)  2.5000000E+01,  2.4000000E+01,  2.3000000E+01,
       (Col 13)  2.2000000E+01,  2.1000000E+01,  2.0000000E+01,
       (Col 16)  1.9000000E+01,  1.8000000E+01,  1.7000000E+01,
       (Col 19)  1.6000000E+01,  1.5000000E+01,  1.4000000E+01,
       (Col 22)  1.3000000E+01,  1.2000000E+01,  1.1000000E+01,
       (Col 25)  1.0000000E+01,  9.0000000E+00,  8.0000000E+00,
       (Col 28)  7.0000000E+00,  6.0000000E+00,  5.0000000E+00,
       (Col 31)  4.0000000E+00,  3.0000000E+00
(Row 03 Col 01)  3.5000000E+01,  3.4000000E+01,  3.3000000E+01,
       (Col 04)  3.2000000E+01,  3.1000000E+01,  3.0000000E+01,
       (Col 07)  2.9000000E+01,  2.8000000E+01,  2.7000000E+01,
       (Col 10)  2.6000000E+01,  2.5000000E+01,  2.4000000E+01,
       (Col 13)  2.3000000E+01,  2.2000000E+01,  2.1000000E+01,
       (Col 16)  2.0000000E+01,  1.9000000E+01,  1.8000000E+01,
       (Col 19)  1.7000000E+01,  1.6000000E+01,  1.5000000E+01,
       (Col 22)  1.4000000E+01,  1.3000000E+01,  1.2000000E+01,
       (Col 25)  1.1000000E+01,  1.0000000E+01,  9.0000000E+00,
       (Col 28)  8.0000000E+00,  7.0000000E+00,  6.0000000E+00,
       (Col 31)  5.0000000E+00,  4.0000000E+00

       .
       .
       .
```

```
(Row 30 Col 01)  6.2000000E+01,  6.1000000E+01,  6.0000000E+01,
        (Col 04)  5.9000000E+01,  5.8000000E+01,  5.7000000E+01,
        (Col 07)  5.6000000E+01,  5.5000000E+01,  5.4000000E+01,
        (Col 10)  5.3000000E+01,  5.2000000E+01,  5.1000000E+01,
        (Col 13)  5.0000000E+01,  4.9000000E+01,  4.8000000E+01,
        (Col 16)  4.7000000E+01,  4.6000000E+01,  4.5000000E+01,
        (Col 19)  4.4000000E+01,  4.3000000E+01,  4.2000000E+01,
        (Col 22)  4.1000000E+01,  4.0000000E+01,  3.9000000E+01,
        (Col 25)  3.8000000E+01,  3.7000000E+01,  3.6000000E+01,
        (Col 28)  3.5000000E+01,  3.4000000E+01,  3.3000000E+01,
        (Col 31)  3.2000000E+01,  3.1000000E+01
(Row 31 Col 01)  6.3000000E+01,  6.2000000E+01,  6.1000000E+01,
        (Col 04)  6.0000000E+01,  5.9000000E+01,  5.8000000E+01,
        (Col 07)  5.7000000E+01,  5.6000000E+01,  5.5000000E+01,
        (Col 10)  5.4000000E+01,  5.3000000E+01,  5.2000000E+01,
        (Col 13)  5.1000000E+01,  5.0000000E+01,  4.9000000E+01,
        (Col 16)  4.8000000E+01,  4.7000000E+01,  4.6000000E+01,
        (Col 19)  4.5000000E+01,  4.4000000E+01,  4.3000000E+01,
        (Col 22)  4.2000000E+01,  4.1000000E+01,  4.0000000E+01,
        (Col 25)  3.9000000E+01,  3.8000000E+01,  3.7000000E+01,
        (Col 28)  3.6000000E+01,  3.5000000E+01,  3.4000000E+01,
        (Col 31)  3.3000000E+01,  3.2000000E+01
(Row 32 Col 01)  6.4000000E+01,  6.3000000E+01,  6.2000000E+01,
        (Col 04)  6.1000000E+01,  6.0000000E+01,  5.9000000E+01,
        (Col 07)  5.8000000E+01,  5.7000000E+01,  5.6000000E+01,
        (Col 10)  5.5000000E+01,  5.4000000E+01,  5.3000000E+01,
        (Col 13)  5.2000000E+01,  5.1000000E+01,  5.0000000E+01,
        (Col 16)  4.9000000E+01,  4.8000000E+01,  4.7000000E+01,
        (Col 19)  4.6000000E+01,  4.5000000E+01,  4.4000000E+01,
        (Col 22)  4.3000000E+01,  4.2000000E+01,  4.1000000E+01,
        (Col 25)  4.0000000E+01,  3.9000000E+01,  3.8000000E+01,
        (Col 28)  3.7000000E+01,  3.6000000E+01,  3.5000000E+01,
        (Col 31)  3.4000000E+01,  3.3000000E+01
```

11.3 M01_INV_PERMUTE_LV_32

release 1

1 Purpose

M01_INV_PERMUTE_LV_32 permutes the values in a long vector of 4-byte values using an INTEGER*4 long vector key. The result is written to a new long vector; the original data is unaffected. The data shuffling implemented is:

ANSWER (KEY(I)) = START (I), I = 1, 1024

using long vector indexing. Hence the key long vector must contain values in the range 1 – 1024, but the values need not be distinct.

2 Specification

```
SUBROUTINE M01_INV_PERMUTE_LV_32(ANSWER , START , KEY)
INTEGER*4 or REAL*4 ANSWER(,) , START(,)
INTEGER*4 KEY(,)
```

3 Description

Local copies of the data and answer long vectors are made, and converted to vector mode. The keys are copied and changed to zero-based offsets, and converted to vector mode. Each row of this key vector set then contains an index of a row in the destination vector set. The data rows are processed in turn and the contents of the addressed row are copied to the (copy of the) destination vector set, indexed by the value in the same row position of the key row. This result vector set is then copied to the answer long vector and converted to matrix mode.

4 References

None

5 Arguments

ANSWER – INTEGER*4 or REAL*4 MATRIX

On exit, ANSWER contains the shuffled version of the input matrix START.

START – INTEGER*4 or REAL*4 MATRIX

On entry, START should contain the data to be shuffled; START is unchanged on exit.

KEY – INTEGER*4 MATRIX

On entry, KEY should contain values in the range 1 – 1024 (not necessarily distinct) describing the required shuffle; KEY is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references routines Z_M01_PLV_CONV_ONLY and Z_M01_PLV_COPY_AND_CONV from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

Because of the way that the routine is coded, you should not assume that the start and key long vectors are processed with an index that increases in a simple way.

10 Keywords

Data movement, permutation, rearrange data, shuffle.

11 Example

The following FORTRAN-PLUS fragment reverses a long vector of integer values.

```
      ENTRY SUBROUTINE ENT
      INTEGER DATA(,), KEY(,), RESULT(,)
      DO 10 I = 1, 1024
      DATA(I) = 3 * I
   10 KEY(I) = 1025 -- I
      CALL M01_PERMUTE_LV_32(RESULT, DATA, KEY)
      TRACE 1 (RESULT)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 7

Integer Matrix Local Variable RESULT in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)        3072,        2976,        2880,        2784,
       (Col 05)        2688,        2592,        2496,        2400,
       (Col 09)        2304,        2208,        2112,        2016,
       (Col 13)        1920,        1824,        1728,        1632,
       (Col 17)        1536,        1440,        1344,        1248,
       (Col 21)        1152,        1056,         960,         864,
       (Col 25)         768,         672,         576,         480,
       (Col 29)         384,         288,         192,          96
(Row 02 Col 01)        3069,        2973,        2877,        2781,
       (Col 05)        2685,        2589,        2493,        2397,
       (Col 09)        2301,        2205,        2109,        2013,
       (Col 13)        1917,        1821,        1725,        1629,
       (Col 17)        1533,        1437,        1341,        1245,
       (Col 21)        1149,        1053,         957,         861,
       (Col 25)         765,         669,         573,         477,
       (Col 29)         381,         285,         189,          93
```

```
(Row 03 Col 01)     3066,      2970,      2874,      2778,
        (Col 05)    2682,      2586,      2490,      2394,
        (Col 09)    2298,      2202,      2106,      2010,
        (Col 13)    1914,      1818,      1722,      1626,
        (Col 17)    1530,      1434,      1338,      1242,
        (Col 21)    1146,      1050,       954,       858,
        (Col 25)     762,       666,       570,       474,
        (Col 29)     378,       282,       186,        90

        .
        .
        .

(Row 30 Col 01)     2985,      2889,      2793,      2697,
        (Col 05)    2601,      2505,      2409,      2313,
        (Col 09)    2217,      2121,      2025,      1929,
        (Col 13)    1833,      1737,      1641,      1545,
        (Col 17)    1449,      1353,      1257,      1161,
        (Col 21)    1065,       969,       873,       777,
        (Col 25)     681,       585,       489,       393,
        (Col 29)     297,       201,       105,         9
(Row 31 Col 01)     2982,      2886,      2790,      2694,
        (Col 05)    2598,      2502,      2406,      2310,
        (Col 09)    2214,      2118,      2022,      1926,
        (Col 13)    1830,      1734,      1638,      1542,
        (Col 17)    1446,      1350,      1254,      1158,
        (Col 21)    1062,       966,       870,       774,
        (Col 25)     678,       582,       486,       390,
        (Col 29)     294,       198,       102,         6
(Row 32 Col 01)     2979,      2883,      2787,      2691,
        (Col 05)    2595,      2499,      2403,      2307,
        (Col 09)    2211,      2115,      2019,      1923,
        (Col 13)    1827,      1731,      1635,      1539,
        (Col 17)    1443,      1347,      1251,      1155,
        (Col 21)    1059,       963,       867,       771,
        (Col 25)     675,       579,       483,       387,
        (Col 29)     291,       195,        99,         3
```

11.4 M01_INV_PERMUTE_ROWS release 1

1 Purpose

M01_INV_PERMUTE_ROWS permutes the first M rows of a matrix according to a permutation vector (IV). The result is equivalent to the FORTRAN-PLUS statements:

```
    DO 10 I = 1, M
10  A_PERMUTED (IV (I), ) = A(I, )
```

2 Specification

```
SUBROUTINE M01_INV_PERMUTE_ROWS(A , AP , IV , N , M)
INTEGER IV ( ) , N , M
<any type> A (,) , AP (,)
```

3 Description

Rows are permuted according to the integer index vector IV such that row I is moved to row IV(I).

4 References

None

5 Arguments

A – <*any type*> MATRIX

On entry, A should contain the matrix whose rows are to be permuted. A may be of any type and is unchanged on exit.

AP – <*any type*> MATRIX

On exit, AP contains the rows of A permuted according to the index vector IV. AP should usually be of the same type as A. If M is less than 32, rows M+1 to 32 are unchanged on exit.

IV – INTEGER VECTOR

On entry, IV should contain the required permutation; that is, row I of A will be moved to row IV(I) of AP. Elements 1 to M of IV must be in the range 1 to 32. If the entries of IV are not all distinct – for example, if IV(I) = IV(J) with J > I – then row AP(IV(J),) will have the value A(J,) on exit. IV is unchanged on exit.

N – INTEGER

On entry, N contains the number of planes in the matrix to be permuted; possible values for N are:

N = 1	for permuting a logical matrix
N = 8	for permuting a character matrix
N = 8*n	for permuting an INTEGER*n or REAL*n matrix

N should be less than 257, and is unchanged on exit.

M – INTEGER

On entry M must contain a value in the range 1 to 32. Only the first M index values of IV are used. M is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_M01_AUX.

8 Accuracy

Not applicable

9 Further Comments

The parameters given as A and AP may be single arrays or part of a matrix set. For example, in:

CALL M01_INV_PERMUTE_COLS (L(,,5), LL(,,10), IV, 1, 32)

L and LL are logical matrix sets of size (at least) 5 and 10 respectively.

You must not use a common block with the names of CZ_M01_HEX1F or CZ_M01_REVERSE.

10 Keywords

Permutation

11 Example

The following FORTRAN-PLUS fragment reverses the order of the rows or a real matrix, that is AP = REVC(A)

```
      ENTRY SUBROUTINE ENT
      REAL A(,), AP(,)
      INTEGER IV()
      DO 10 I = 1, 32
   10 IV(I) = 33 - I
      DO 20 I = 1, 32
      DO 20 J = 1, 32
   20 A(I,J) = FLOAT (I + J)
      CALL M01_INV_PERMUTE_ROWS (A, AP, IV, 32, 32)
      TRACE 1 (AP)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 10

Real Matrix Local Variable AP in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)  3.3000000E+01,  3.4000000E+01,  3.5000000E+01,
       (Col 04)  3.6000000E+01,  3.7000000E+01,  3.8000000E+01,
       (Col 07)  3.9000000E+01,  4.0000000E+01,  4.1000000E+01,
       (Col 10)  4.2000000E+01,  4.3000000E+01,  4.4000000E+01,
       (Col 13)  4.5000000E+01,  4.6000000E+01,  4.7000000E+01,
       (Col 16)  4.8000000E+01,  4.9000000E+01,  5.0000000E+01,
       (Col 19)  5.1000000E+01,  5.2000000E+01,  5.3000000E+01,
       (Col 22)  5.4000000E+01,  5.5000000E+01,  5.6000000E+01,
       (Col 25)  5.7000000E+01,  5.8000000E+01,  5.9000000E+01,
       (Col 28)  6.0000000E+01,  6.1000000E+01,  6.2000000E+01,
       (Col 31)  6.3000000E+01,  6.4000000E+01
(Row 02 Col 01)  3.2000000E+01,  3.3000000E+01,  3.4000000E+01,
       (Col 04)  3.5000000E+01,  3.6000000E+01,  3.7000000E+01,
       (Col 07)  3.8000000E+01,  3.9000000E+01,  4.0000000E+01,
       (Col 10)  4.1000000E+01,  4.2000000E+01,  4.3000000E+01,
       (Col 13)  4.4000000E+01,  4.5000000E+01,  4.6000000E+01,
       (Col 16)  4.7000000E+01,  4.8000000E+01,  4.9000000E+01,
       (Col 19)  5.0000000E+01,  5.1000000E+01,  5.2000000E+01,
       (Col 22)  5.3000000E+01,  5.4000000E+01,  5.5000000E+01,
       (Col 25)  5.6000000E+01,  5.7000000E+01,  5.8000000E+01,
       (Col 28)  5.9000000E+01,  6.0000000E+01,  6.1000000E+01,
       (Col 31)  6.2000000E+01,  6.3000000E+01
(Row 03 Col 01)  3.1000000E+01,  3.2000000E+01,  3.3000000E+01,
       (Col 04)  3.4000000E+01,  3.5000000E+01,  3.6000000E+01,
       (Col 07)  3.7000000E+01,  3.8000000E+01,  3.9000000E+01,
       (Col 10)  4.0000000E+01,  4.1000000E+01,  4.2000000E+01,
       (Col 13)  4.3000000E+01,  4.4000000E+01,  4.5000000E+01,
       (Col 16)  4.6000000E+01,  4.7000000E+01,  4.8000000E+01,
       (Col 19)  4.9000000E+01,  5.0000000E+01,  5.1000000E+01,
       (Col 22)  5.2000000E+01,  5.3000000E+01,  5.4000000E+01,
       (Col 25)  5.5000000E+01,  5.6000000E+01,  5.7000000E+01,
       (Col 28)  5.8000000E+01,  5.9000000E+01,  6.0000000E+01,
       (Col 31)  6.1000000E+01,  6.2000000E+01

       .
       .
       .
```

```
(Row 30 Col 01)  4.0000000E+00,  5.0000000E+00,  6.0000000E+00,
        (Col 04)  7.0000000E+00,  8.0000000E+00,  9.0000000E+00,
        (Col 07)  1.0000000E+01,  1.1000000E+01,  1.2000000E+01,
        (Col 10)  1.3000000E+01,  1.4000000E+01,  1.5000000E+01,
        (Col 13)  1.6000000E+01,  1.7000000E+01,  1.8000000E+01,
        (Col 16)  1.9000000E+01,  2.0000000E+01,  2.1000000E+01,
        (Col 19)  2.2000000E+01,  2.3000000E+01,  2.4000000E+01,
        (Col 22)  2.5000000E+01,  2.6000000E+01,  2.7000000E+01,
        (Col 25)  2.8000000E+01,  2.9000000E+01,  3.0000000E+01,
        (Col 28)  3.1000000E+01,  3.2000000E+01,  3.3000000E+01,
        (Col 31)  3.4000000E+01,  3.5000000E+01
(Row 31 Col 01)  3.0000000E+00,  4.0000000E+00,  5.0000000E+00,
        (Col 04)  6.0000000E+00,  7.0000000E+00,  8.0000000E+00,
        (Col 07)  9.0000000E+00,  1.0000000E+01,  1.1000000E+01,
        (Col 10)  1.2000000E+01,  1.3000000E+01,  1.4000000E+01,
        (Col 13)  1.5000000E+01,  1.6000000E+01,  1.7000000E+01,
        (Col 16)  1.8000000E+01,  1.9000000E+01,  2.0000000E+01,
        (Col 19)  2.1000000E+01,  2.2000000E+01,  2.3000000E+01,
        (Col 22)  2.4000000E+01,  2.5000000E+01,  2.6000000E+01,
        (Col 25)  2.7000000E+01,  2.8000000E+01,  2.9000000E+01,
        (Col 28)  3.0000000E+01,  3.1000000E+01,  3.2000000E+01,
        (Col 31)  3.3000000E+01,  3.4000000E+01
(Row 32 Col 01)  2.0000000E+00,  3.0000000E+00,  4.0000000E+00,
        (Col 04)  5.0000000E+00,  6.0000000E+00,  7.0000000E+00,
        (Col 07)  8.0000000E+00,  9.0000000E+00,  1.0000000E+01,
        (Col 10)  1.1000000E+01,  1.2000000E+01,  1.3000000E+01,
        (Col 13)  1.4000000E+01,  1.5000000E+01,  1.6000000E+01,
        (Col 16)  1.7000000E+01,  1.8000000E+01,  1.9000000E+01,
        (Col 19)  2.0000000E+01,  2.1000000E+01,  2.2000000E+01,
        (Col 22)  2.3000000E+01,  2.4000000E+01,  2.5000000E+01,
        (Col 25)  2.6000000E+01,  2.7000000E+01,  2.8000000E+01,
        (Col 28)  2.9000000E+01,  3.0000000E+01,  3.1000000E+01,
        (Col 31)  3.2000000E+01,  3.3000000E+01
```

11.5 M01_PERMUTE_COLS

release 1

1 Purpose

M01_PERMUTE_COLS permutes the first M columns of a matrix according to a permutation vector(IV). The result is equivalent to the FORTRAN-PLUS statements:

```
      DO 10 I = 1, M
   10 A_PERMUTED(,I) = A(,IV(I))
```

2 Specification

```
SUBROUTINE M01_PERMUTE_COLS(A , AP , IV , N , M)
INTEGER IV() , N , M
<any type> A(,) , AP(,)
```

3 Description

Columns are permuted according to the integer index vector IV, such that column IV(I) is moved to column I.

4 References

None

5 Arguments

A – <*any type*> MATRIX

On entry, A contains the matrix whose columns are to be permuted. A may be of any type, and is unchanged on exit.

AP – <*any type*> MATRIX

On exit, AP contains the columns of A permuted according to the index vector IV. AP should usually be of the same type as A. If M is less than 32, columns M+1 to 32 are unchanged on exit.

IV – INTEGER VECTOR

On entry, IV contains the required permutation, that is column IV(I) of A will be moved to column I of AP. Elements 1 to M of IV must be in the range 1 to 32 (but need not be distinct). IV is unchanged on exit.

N – INTEGER

On entry, N contains the number of planes in the matrix to be permuted; possible values for N are:

N = 1	for permuting a logical matrix
N = 8	for permuting a character matrix
N = 8**n*	for permuting an INTEGER**n* or REAL**n* matrix

N should be less than 257, and is unchanged on exit.

M – INTEGER

On entry M must contain a value in the range 1 to 32. Only the first M index values of IV are used; M is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_M01_AUX.

8 Accuracy

Not applicable

9 Further Comments

The parameters given as A and AP may be single arrays or part of a matrix set. For example, in:

CALL M01_PERMUTE_COLS (L(,,5),LL(,,10),IV,1,32)

L and LL are logical matrix sets of size (at least) 5 and 10 respectively.

You must not use a common block with the name of CZ_M01_HEX1F.

10 Keywords

Permutation

11 Example

The following FORTRAN-PLUS fragment reverses the order of the columns of a real matrix, that is, AP = REVC(A).

```
      ENTRY SUBROUTINE ENT
      REAL A(,), AP(,)
      INTEGER IV()
      DO  10   I = 1,32
   10 IV(I) = 33 - I
      DO 20 J = 1, 32
      DO 20 I = 1, 32
   20 A(I,J) = FLOAT (I + J)
      CALL M01_PERMUTE_COLS(A, AP, IV, 32, 32)
      TRACE 1 (AP)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 10

Real Matrix Local Variable AP in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)  3.3000000E+01,  3.2000000E+01,  3.1000000E+01,
       (Col 04)  3.0000000E+01,  2.9000000E+01,  2.8000000E+01,
       (Col 07)  2.7000000E+01,  2.6000000E+01,  2.5000000E+01,
       (Col 10)  2.4000000E+01,  2.3000000E+01,  2.2000000E+01,
       (Col 13)  2.1000000E+01,  2.0000000E+01,  1.9000000E+01,
       (Col 16)  1.8000000E+01,  1.7000000E+01,  1.6000000E+01,
       (Col 19)  1.5000000E+01,  1.4000000E+01,  1.3000000E+01,
       (Col 22)  1.2000000E+01,  1.1000000E+01,  1.0000000E+01,
       (Col 25)  9.0000000E+00,  8.0000000E+00,  7.0000000E+00,
       (Col 28)  6.0000000E+00,  5.0000000E+00,  4.0000000E+00,
       (Col 31)  3.0000000E+00,  2.0000000E+00
(Row 02 Col 01)  3.4000000E+01,  3.3000000E+01,  3.2000000E+01,
       (Col 04)  3.1000000E+01,  3.0000000E+01,  2.9000000E+01,
       (Col 07)  2.8000000E+01,  2.7000000E+01,  2.6000000E+01,
       (Col 10)  2.5000000E+01,  2.4000000E+01,  2.3000000E+01,
       (Col 13)  2.2000000E+01,  2.1000000E+01,  2.0000000E+01,
       (Col 16)  1.9000000E+01,  1.8000000E+01,  1.7000000E+01,
       (Col 19)  1.6000000E+01,  1.5000000E+01,  1.4000000E+01,
       (Col 22)  1.3000000E+01,  1.2000000E+01,  1.1000000E+01,
       (Col 25)  1.0000000E+01,  9.0000000E+00,  8.0000000E+00,
       (Col 28)  7.0000000E+00,  6.0000000E+00,  5.0000000E+00,
       (Col 31)  4.0000000E+00,  3.0000000E+00
(Row 03 Col 01)  3.5000000E+01,  3.4000000E+01,  3.3000000E+01,
       (Col 04)  3.2000000E+01,  3.1000000E+01,  3.0000000E+01,
       (Col 07)  2.9000000E+01,  2.8000000E+01,  2.7000000E+01,
       (Col 10)  2.6000000E+01,  2.5000000E+01,  2.4000000E+01,
       (Col 13)  2.3000000E+01,  2.2000000E+01,  2.1000000E+01,
       (Col 16)  2.0000000E+01,  1.9000000E+01,  1.8000000E+01,
       (Col 19)  1.7000000E+01,  1.6000000E+01,  1.5000000E+01,
       (Col 22)  1.4000000E+01,  1.3000000E+01,  1.2000000E+01,
       (Col 25)  1.1000000E+01,  1.0000000E+01,  9.0000000E+00,
       (Col 28)  8.0000000E+00,  7.0000000E+00,  6.0000000E+00,
       (Col 31)  5.0000000E+00,  4.0000000E+00

       .
       .
       .
```

```
(Row 30 Col 01)  6.2000000E+01,  6.1000000E+01,  6.0000000E+01,
        (Col 04)  5.9000000E+01,  5.8000000E+01,  5.7000000E+01,
        (Col 07)  5.6000000E+01,  5.5000000E+01,  5.4000000E+01,
        (Col 10)  5.3000000E+01,  5.2000000E+01,  5.1000000E+01,
        (Col 13)  5.0000000E+01,  4.9000000E+01,  4.8000000E+01,
        (Col 16)  4.7000000E+01,  4.6000000E+01,  4.5000000E+01,
        (Col 19)  4.4000000E+01,  4.3000000E+01,  4.2000000E+01,
        (Col 22)  4.1000000E+01,  4.0000000E+01,  3.9000000E+01,
        (Col 25)  3.8000000E+01,  3.7000000E+01,  3.6000000E+01,
        (Col 28)  3.5000000E+01,  3.4000000E+01,  3.3000000E+01,
        (Col 31)  3.2000000E+01,  3.1000000E+01
(Row 31 Col 01)  6.3000000E+01,  6.2000000E+01,  6.1000000E+01,
        (Col 04)  6.0000000E+01,  5.9000000E+01,  5.8000000E+01,
        (Col 07)  5.7000000E+01,  5.6000000E+01,  5.5000000E+01,
        (Col 10)  5.4000000E+01,  5.3000000E+01,  5.2000000E+01,
        (Col 13)  5.1000000E+01,  5.0000000E+01,  4.9000000E+01,
        (Col 16)  4.8000000E+01,  4.7000000E+01,  4.6000000E+01,
        (Col 19)  4.5000000E+01,  4.4000000E+01,  4.3000000E+01,
        (Col 22)  4.2000000E+01,  4.1000000E+01,  4.0000000E+01,
        (Col 25)  3.9000000E+01,  3.8000000E+01,  3.7000000E+01,
        (Col 28)  3.6000000E+01,  3.5000000E+01,  3.4000000E+01,
        (Col 31)  3.3000000E+01,  3.2000000E+01
(Row 32 Col 01)  6.4000000E+01,  6.3000000E+01,  6.2000000E+01,
        (Col 04)  6.1000000E+01,  6.0000000E+01,  5.9000000E+01,
        (Col 07)  5.8000000E+01,  5.7000000E+01,  5.6000000E+01,
        (Col 10)  5.5000000E+01,  5.4000000E+01,  5.3000000E+01,
        (Col 13)  5.2000000E+01,  5.1000000E+01,  5.0000000E+01,
        (Col 16)  4.9000000E+01,  4.8000000E+01,  4.7000000E+01,
        (Col 19)  4.6000000E+01,  4.5000000E+01,  4.4000000E+01,
        (Col 22)  4.3000000E+01,  4.2000000E+01,  4.1000000E+01,
        (Col 25)  4.0000000E+01,  3.9000000E+01,  3.8000000E+01,
        (Col 28)  3.7000000E+01,  3.6000000E+01,  3.5000000E+01,
        (Col 31)  3.4000000E+01,  3.3000000E+01
```

11.6 M01_PERMUTE_LV_32

release 1

1 Purpose

M01_PERMUTE_LV_32 permutes the values in a long vector of 4-byte values using an INTEGER*4 long vector key. The result is written to a new long vector and the original data is unaffected. The data shuffling implemented is:

ANSWER (I) = START (KEY(I)), I = 1, 1024

using long vector indexing. Hence the key long vector must contain values in the range 1 – 1024, but the values need not be distinct.

2 Specification

```
SUBROUTINE M01_PERMUTE_LV_32(ANSWER , START , KEY)
INTEGER*4 or REAL*4 ANSWER(,) , START(,)
INTEGER*4 KEY(,)
```

3 Description

A local copy of the data is made, and converted to vector mode. The keys are copied and changed to zero-based offsets, then converted to vector mode. Each row of this key vector set then contains an index of a row in the data vector set. The key rows are processed in turn and the contents of the addressed row are copied to another vector set in the same row position as the key row. This result vector set is then copied to the answer long vector, and converted to matrix mode.

4 References

None

5 Arguments

ANSWER – INTEGER*4 or REAL*4 MATRIX

On exit, ANSWER contains the shuffled version of the input matrix START.

START – INTEGER*4 or REAL*4 MATRIX

On entry, START should contain the data to be shuffled; START is unchanged on exit.

KEY – INTEGER*4 MATRIX

On entry, KEY should contain values in the range 1 – 1024 (not necessarily distinct) describing the required shuffle; KEY is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

This routine references routines Z_M01_PLV_CONV_ONLY and Z_M01_PLV_COPY_AND_CONV from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data movement, permutation, rearrange data, shuffle

11 Example

The following FORTRAN-PLUS fragment reverses a long vector of integer values.

```
      ENTRY SUBROUTINE ENT
      INTEGER DATA(,), KEY(,), RESULT(,)
      DO 10 I = 1, 1024
      DATA(I) = 3 * I
   10 KEY(I) = 1025 - I
      CALL M01_PERMUTE_LV_32(RESULT, DATA, KEY)
      TRACE 1 (RESULT)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 7

Integer Matrix Local Variable RESULT in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)       3072,        2976,        2880,        2784,
       (Col 05)       2688,        2592,        2496,        2400,
       (Col 09)       2304,        2208,        2112,        2016,
       (Col 13)       1920,        1824,        1728,        1632,
       (Col 17)       1536,        1440,        1344,        1248,
       (Col 21)       1152,        1056,         960,         864,
       (Col 25)        768,         672,         576,         480,
       (Col 29)        384,         288,         192,          96
(Row 02 Col 01)       3069,        2973,        2877,        2781,
       (Col 05)       2685,        2589,        2493,        2397,
       (Col 09)       2301,        2205,        2109,        2013,
       (Col 13)       1917,        1821,        1725,        1629,
       (Col 17)       1533,        1437,        1341,        1245,
       (Col 21)       1149,        1053,         957,         861,
       (Col 25)        765,         669,         573,         477,
       (Col 29)        381,         285,         189,          93
```

```
(Row 03 Col 01)      3066,      2970,      2874,      2778,
        (Col 05)     2682,      2586,      2490,      2394,
        (Col 09)     2298,      2202,      2106,      2010,
        (Col 13)     1914,      1818,      1722,      1626,
        (Col 17)     1530,      1434,      1338,      1242,
        (Col 21)     1146,      1050,       954,       858,
        (Col 25)      762,       666,       570,       474,
        (Col 29)      378,       282,       186,        90

        .
        .
        .

(Row 30 Col 01)      2985,      2889,      2793,      2697,
        (Col 05)     2601,      2505,      2409,      2313,
        (Col 09)     2217,      2121,      2025,      1929,
        (Col 13)     1833,      1737,      1641,      1545,
        (Col 17)     1449,      1353,      1257,      1161,
        (Col 21)     1065,       969,       873,       777,
        (Col 25)      681,       585,       489,       393,
        (Col 29)      297,       201,       105,         9
(Row 31 Col 01)      2982,      2886,      2790,      2694,
        (Col 05)     2598,      2502,      2406,      2310,
        (Col 09)     2214,      2118,      2022,      1926,
        (Col 13)     1830,      1734,      1638,      1542,
        (Col 17)     1446,      1350,      1254,      1158,
        (Col 21)     1062,       966,       870,       774,
        (Col 25)      678,       582,       486,       390,
        (Col 29)      294,       198,       102,         6
(Row 32 Col 01)      2979,      2883,      2787,      2691,
        (Col 05)     2595,      2499,      2403,      2307,
        (Col 09)     2211,      2115,      2019,      1923,
        (Col 13)     1827,      1731,      1635,      1539,
        (Col 17)     1443,      1347,      1251,      1155,
        (Col 21)     1059,       963,       867,       771,
        (Col 25)      675,       579,       483,       387,
        (Col 29)      291,       195,        99,         3
```

11.7 M01_PERMUTE_ROWS

release 1

1 Purpose

M01_PERMUTE_ROWS permutes the first M rows of a matrix according to a permutation vector (IV). The result is equivalent to the FORTRAN-PLUS statements:

```
      DO 10 I = 1, M
10    10 A_PERMUTED(I,) = A(IV(I),)
```

2 Specification

```
SUBROUTINE M01_PERMUTE_ROWS(A , AP , IV , N , M)
INTEGER IV() , N , M
<any type> A(,) , AP(,)
```

3 Description

Rows are permuted according to the integer index vector IV such that row IV(I) is moved to row I.

4 References

None

5 Arguments

A – <*any type*> MATRIX

On entry, A should contain the matrix whose rows are to be permuted. A may be of any type and is unchanged on exit.

AP – <*any type*> MATRIX

On exit, AP contains the rows of A permuted according to the index vector IV. AP should usually be of the same type as A. If M is less than 32, rows M+1 to 32 are unchanged on exit.

IV – INTEGER VECTOR

On entry, IV should contain the required permutation; that is, row I of A will be moved to row IV(I) of AP. Elements 1 to M of IV must be in the range 1 to 32. If the entries of IV are not all distinct – for example, if IV(I) = IV(J) with J > I – then row AP (IV(J),) will have the value A(J,) on exit. IV is unchanged on exit.

N – INTEGER

On entry, N contains the number of planes in the matrix to be permuted; possible values for N are:

N = 1	for permuting a logical matrix
N = 8	for permuting a character matrix
N = 8*n	for permuting an INTEGER*n or REAL*n matrix

N should be less than 257, and is unchanged on exit.

M – INTEGER

On entry M must contain a value in the range 1 to 32. Only the first M index values of IV are used. M is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_M01_AUX.

8 Accuracy

Not applicable

9 Further Comments

The parameter given as A and AP may be single arrays or part of a matrix set. For example, in:

CALL M01_PERMUTE_ROWS(L(,,5),LL(,,10),IV,1,32)

L and LL are logical matrix sets of size (at least) 5 and 10 respectively.

You must not use common blocks with name CZ_M01_HEX1F.

10 Keywords

Permutation

11 Example

The following FORTRAN-PLUS fragment given reverses the order of the rows of a real matrix that is, AP = REVC(A).

```
      ENTRY SUBROUTINE ENT
      REAL A(,), AP (,)
      INTEGER IV()
      DO 10 I = 1,32
   10 IV (I) = 33 - I
      DO 20 I = 1, 32
      DO 20 J = 1, 32
   20 A(I,J) = FLOAT(I + J)
      CALL M01_PERMUTE_ROWS (A, AP, IV, 32, 32)
      TRACE 1 (AP)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 10

Real Matrix Local Variable AP in 32 bits -- addressed by Stack + 0.10

(Row 01 Col 01)  3.3000000E+01,  3.4000000E+01,  3.5000000E+01,
       (Col 04)  3.6000000E+01,  3.7000000E+01,  3.8000000E+01,
       (Col 07)  3.9000000E+01,  4.0000000E+01,  4.1000000E+01,
       (Col 10)  4.2000000E+01,  4.3000000E+01,  4.4000000E+01,
       (Col 13)  4.5000000E+01,  4.6000000E+01,  4.7000000E+01,
       (Col 16)  4.8000000E+01,  4.9000000E+01,  5.0000000E+01,
       (Col 19)  5.1000000E+01,  5.2000000E+01,  5.3000000E+01,
       (Col 22)  5.4000000E+01,  5.5000000E+01,  5.6000000E+01,
       (Col 25)  5.7000000E+01,  5.8000000E+01,  5.9000000E+01,
       (Col 28)  6.0000000E+01,  6.1000000E+01,  6.2000000E+01,
       (Col 31)  6.3000000E+01,  6.4000000E+01
(Row 02 Col 01)  3.2000000E+01,  3.3000000E+01,  3.4000000E+01,
       (Col 04)  3.5000000E+01,  3.6000000E+01,  3.7000000E+01,
       (Col 07)  3.8000000E+01,  3.9000000E+01,  4.0000000E+01,
       (Col 10)  4.1000000E+01,  4.2000000E+01,  4.3000000E+01,
       (Col 13)  4.4000000E+01,  4.5000000E+01,  4.6000000E+01,
       (Col 16)  4.7000000E+01,  4.8000000E+01,  4.9000000E+01,
       (Col 19)  5.0000000E+01,  5.1000000E+01,  5.2000000E+01,
       (Col 22)  5.3000000E+01,  5.4000000E+01,  5.5000000E+01,
       (Col 25)  5.6000000E+01,  5.7000000E+01,  5.8000000E+01,
       (Col 28)  5.9000000E+01,  6.0000000E+01,  6.1000000E+01,
       (Col 31)  6.2000000E+01,  6.3000000E+01
(Row 03 Col 01)  3.1000000E+01,  3.2000000E+01,  3.3000000E+01,
       (Col 04)  3.4000000E+01,  3.5000000E+01,  3.6000000E+01,
       (Col 07)  3.7000000E+01,  3.8000000E+01,  3.9000000E+01,
       (Col 10)  4.0000000E+01,  4.1000000E+01,  4.2000000E+01,
       (Col 13)  4.3000000E+01,  4.4000000E+01,  4.5000000E+01,
       (Col 16)  4.6000000E+01,  4.7000000E+01,  4.8000000E+01,
       (Col 19)  4.9000000E+01,  5.0000000E+01,  5.1000000E+01,
       (Col 22)  5.2000000E+01,  5.3000000E+01,  5.4000000E+01,
       (Col 25)  5.5000000E+01,  5.6000000E+01,  5.7000000E+01,
       (Col 28)  5.8000000E+01,  5.9000000E+01,  6.0000000E+01,
       (Col 31)  6.1000000E+01,  6.2000000E+01

         .
         .
         .
```

```
(Row 30 Col 01)  4.0000000E+00,  5.0000000E+00,  6.0000000E+00,
        (Col 04)  7.0000000E+00,  8.0000000E+00,  9.0000000E+00,
        (Col 07)  1.0000000E+01,  1.1000000E+01,  1.2000000E+01,
        (Col 10)  1.3000000E+01,  1.4000000E+01,  1.5000000E+01,
        (Col 13)  1.6000000E+01,  1.7000000E+01,  1.8000000E+01,
        (Col 16)  1.9000000E+01,  2.0000000E+01,  2.1000000E+01,
        (Col 19)  2.2000000E+01,  2.3000000E+01,  2.4000000E+01,
        (Col 22)  2.5000000E+01,  2.6000000E+01,  2.7000000E+01,
        (Col 25)  2.8000000E+01,  2.9000000E+01,  3.0000000E+01,
        (Col 28)  3.1000000E+01,  3.2000000E+01,  3.3000000E+01,
        (Col 31)  3.4000000E+01,  3.5000000E+01
(Row 31 Col 01)  3.0000000E+00,  4.0000000E+00,  5.0000000E+00,
        (Col 04)  6.0000000E+00,  7.0000000E+00,  8.0000000E+00,
        (Col 07)  9.0000000E+00,  1.0000000E+01,  1.1000000E+01,
        (Col 10)  1.2000000E+01,  1.3000000E+01,  1.4000000E+01,
        (Col 13)  1.5000000E+01,  1.6000000E+01,  1.7000000E+01,
        (Col 16)  1.8000000E+01,  1.9000000E+01,  2.0000000E+01,
        (Col 19)  2.1000000E+01,  2.2000000E+01,  2.3000000E+01,
        (Col 22)  2.4000000E+01,  2.5000000E+01,  2.6000000E+01,
        (Col 25)  2.7000000E+01,  2.8000000E+01,  2.9000000E+01,
        (Col 28)  3.0000000E+01,  3.1000000E+01,  3.2000000E+01,
        (Col 31)  3.3000000E+01,  3.4000000E+01
(Row 32 Col 01)  2.0000000E+00,  3.0000000E+00,  4.0000000E+00,
        (Col 04)  5.0000000E+00,  6.0000000E+00,  7.0000000E+00,
        (Col 07)  8.0000000E+00,  9.0000000E+00,  1.0000000E+01,
        (Col 10)  1.1000000E+01,  1.2000000E+01,  1.3000000E+01,
        (Col 13)  1.4000000E+01,  1.5000000E+01,  1.6000000E+01,
        (Col 16)  1.7000000E+01,  1.8000000E+01,  1.9000000E+01,
        (Col 19)  2.0000000E+01,  2.1000000E+01,  2.2000000E+01,
        (Col 22)  2.3000000E+01,  2.4000000E+01,  2.5000000E+01,
        (Col 25)  2.6000000E+01,  2.7000000E+01,  2.8000000E+01,
        (Col 28)  2.9000000E+01,  3.0000000E+01,  3.1000000E+01,
        (Col 31)  3.2000000E+01,  3.3000000E+01
```

11.8 M01_SORT_V_I4

release 1

1 Purpose

M01_SORT_V_I4 sorts the first N elements of an integer vector into ascending or descending order. The permutation required to perform the sort is returned to the calling routine.

2 Specification

```
SUBROUTINE M01_SORT_V_I4 (IV , N , UP , PERM , IFAIL)
INTEGER *1 PERM ( )
INTEGER IV( ) , N , IFAIL
LOGICAL UP
```

3 Description

The sort is carried out by spreading the vector, IV, across the DAP and counting the number of elements less than or equal to each particular element. Comparing this count with an index vector and selecting the relevant element from each column of the DAP completes the sort when all elements of IV are distinct. If there are repeated elements in IV, a $\log_2$ duplication process is carried out to regenerate the multiple values.

4 References

None

5 Arguments

IV – INTEGER VECTOR

On entry, components 1 to N of IV contain the elements to be sorted. On exit, components 1 to N will have been sorted as required. Elements N+1 to 32 are unchanged on exit.

N – INTEGER

On entry, N specifies how many components of IV are to be sorted. N must lie in the range 1 to 32, and is unchanged on exit.

UP – LOGICAL

If UP is .TRUE. on entry, then IV is sorted into ascending order, otherwise IV is sorted into descending order. UP is unchanged on exit.

PERM – INTEGER *1 VECTOR

On exit, PERM contains the permutation required to perform the sort, that is, the sort was equivalent to:

```
     DO 10 I = 1, N
10   JV(I) = IV(PERM(I))
```

Elements N+1 to 32 of PERM are zero on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1 N is not in the range 1 to 32

7 Auxiliary Routines

The routine calls the General Support library routines X05_NORTH_BOUNDARY, X05_PATTERN and X05_SHORT_INDEX.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Sorting

11 Example

The vector to be sorted consists of the numbers 1 to 8, each repeated 4 times. The vector is sorted into ascending order.

Host program

```
      INTEGER IV(32), PERM(32)
      COMMON /VEC1/IV /VEC2/PERM
      COMMON /SCALAR/N,IFAIL

      N = 32
      DO 10 I = 1, 32
   10 IV(I) = MOD(I-1, 8) + 1

      CALL DAPCON('ent.dd')
      CALL DAPSEN('SCALAR',N,1)
      CALL DAPSEN('VEC1',IV,32)

      CALL DAPENT('ENT')

      CALL DAPREC('SCALAR',N,2)
      CALL DAPREC('VEC1',IV,32)
      CALL DAPREC('VEC2',PERM,32)

      CALL DAPREL

      WRITE (6, 100) IFAIL, (IV(I), I = 1,32)
  100 FORMAT ('IFAIL = ', I1, //, 'SORTED DATA', //, (4I5))
      WRITE (6,200) (PERM(I), I = 1,32)
  200 FORMAT (/,'PERMUTATION', //, (4I5))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      INTEGER IV(), PERM4()
      INTEGER *1 PERM()
      COMMON /VEC1/IV  /VEC2/PERM4
      COMMON /SCALAR/N,IFAIL

      CALL CONVFSI(N,1)
      CALL CONVFVI(IV,32,1)

      CALL M01_SORT_V_I4(IV,N, .TRUE., PERM, IFAIL)

      PERM4 = PERM

      CALL CONVVFI(IV,32,1)
      CALL CONVVFI(PERM4,32,1)
      CALL CONVSFI(N,2)

      RETURN
      END
```

Results

```
IFAIL = 0

SORTED DATA

    1    1    1    1
    2    2    2    2
    3    3    3    3
    4    4    4    4
    5    5    5    5
    6    6    6    6
    7    7    7    7
    8    8    8    8

PERMUTATION

    1    1    1    1
    2    2    2    2
    3    3    3    3
    4    4    4    4
    5    5    5    5
    6    6    6    6
    7    7    7    7
    8    8    8    8
```

11.9 M01_SORT_V_R4

release 1

1 Purpose

M01_SORT_V_R4 sorts the first N elements of a real vector into ascending or descending order. The permutation required to perform the sort is returned to the calling routine.

2 Specification

```
SUBROUTINE M01_SORT_V_R4(RV , N , UP , PERM , IFAIL)
INTEGER *1 PERM()
INTEGER N , IFAIL
REAL RV()
LOGICAL UP
```

3 Description

The sort is carried out by spreading the vector RV across the DAP, and counting the number of elements less than or equal to each particular element; comparing this with an index vector and selecting the relevant element from each column of the DAP completes the sort when all elements of RV are distinct. If there are repeated elements in RV, a $\log_2$ duplication process is carried out to regenerate the multiple values.

4 References

None

5 Arguments

RV – REAL VECTOR

On entry, components 1 to N of RV contain the elements to be sorted. On exit, components 1 to N will have been sorted as required. Elements N+1 to 32 are unchanged on exit.

N – INTEGER

On entry N specifies how many components of RV are to be sorted. N must lie in the range 1 to 32, and is unchanged on exit.

UP – LOGICAL

If UP is .TRUE. on entry, then RV is sorted into ascending order, otherwise RV is sorted into descending order. UP is unchanged on exit.

PERM – INTEGER *1 VECTOR

On exit PERM contains the permutation required to perform the sort, that is, the sort was equivalent to:

```
      DO 10 I = 1, N
10    SV(I) = RV(PERM(I))
```

Elements N+1 to 32 of PERM are zero on exit.

5 Arguments – continued

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1 N is not in the range 1 to 32

7 Auxiliary Routines

The routine calls the General Support library routines X05_NORTH_BOUNDARY, X05_PATTERN and X05_SHORT_INDEX.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Sorting

11 Example

The vector to be sorted consists of the numbers 1.0 to 8.0, each repeated 4 times. The vector is sorted into ascending order.

Host program

```
      INTEGER PERM (32)
      REAL RV(32)
      COMMON /VEC1/RV /VEC2/PERM
      COMMON /SCALAR/N,IFAIL

      N = 32
      DO 10 I = 1,32
   10 RV(I) = MOD(I-1,8)+1

      CALL DAPCON('ent.dd')
      CALL DAPSEN('SCALAR',N,1)
      CALL DAPSEN('VEC1',RV,32)

      CALL DAPENT('ENT')
```

```
      CALL DAPREC('SCALAR',N,2)
      CALL DAPREC('VEC1',RV,32)
      CALL DAPREC('VEC2',PERM,32)

      CALL DAPREL

      WRITE (6, 100) IFAIL, (RV(I), I = 1,32)
  100 FORMAT ('IFAIL = ', I1, //, 'SORTED DATA', //, (4F5.0))
      WRITE (6,200) (PERM(I), I = 1,32)
  200 FORMAT (/,'PERMUTATION', //, (4I5))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT

      INTEGER PERM4()
      INTEGER *1 PERM ()
      REAL RV()
      COMMON /VEC1/RV /VEC2/PERM4
      COMMON /SCALAR/ N,IFAIL

      CALL CONVFSI(N,1)
      CALL CONVFVE(RV,32,1)

      CALL M01_SORT_V_R4(RV, N, .TRUE., PERM, IFAIL)

      PERM4 = PERM

      CALL CONVVFE(RV, 32, 1)
      CALL CONVVFI(PERM4, 32, 1)
      CALL CONVSFI(N,2)

      RETURN
      END
```

Results

```
IFAIL = 0

SORTED DATA

    1.    1.    1.    1.
    2.    2.    2.    2.
    3.    3.    3.    3.
    4.    4.    4.    4.
    5.    5.    5.    5.
    6.    6.    6.    6.
    7.    7.    7.    7.
    8.    8.    8.    8.

PERMUTATION

     1     1     1     1
     2     2     2     2
     3     3     3     3
     4     4     4     4
     5     5     5     5
     6     6     6     6
     7     7     7     7
     8     8     8     8
```

Chapter 12

S – Special functions

Contents:

12.1 S04_ARC_COS

release 1

1 Purpose

S04_ARC_COS returns the value of the inverse cosine function arccos (x) for a matrix argument. The result lies in the range $[0, \pi]$.

2 Specification

REAL MATRIX FUNCTION S04_ARC_COS (X , EMASK)
REAL X (,)
LOGICAL EMASK (,)

3 Description

Arccos is approximated using a Tschebyshev polynomial expansion of the form:

$$\arcsin(x) \simeq p(x) = x \sum{}' a_r T_r(t) \qquad \text{where} \qquad t = 4x^2 - 1$$

where $\sum'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum'$ is half the first term in $\sum$.

The approximation for different values of the argument x is as follows:

$$\arccos(x) \simeq \pi/2 - p(x) \qquad \text{for} \qquad x \in [-1/\sqrt{2},\ 1/\sqrt{2})$$

$$\arccos(x) \simeq \pi - p(\sqrt{1-x^2}) \qquad \text{for} \qquad x \in [-1,\ -1/\sqrt{2})$$

$$\arccos(x) \simeq p(\sqrt{1-x^2}) \qquad \text{for} \qquad x \in (1/\sqrt{2},\ 1]$$

For $|x| > 1$ the result is undefined.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 4 section 4, p 79: Dover Publications 1968.

[2] FOX L and PARKER I

Chebyshev Polynomials in Numerical Analysis: Oxford University Press; 1968.

5 Arguments

X – REAL MATRIX

On entry, X contains the points at which the evaluation of arccos is required. All elements of X must be defined on entry. X is unchanged on exit.

EMASK – LOGICAL MATRIX

On exit, EMASK is set .TRUE. at positions corresponding to invalid arguments (see **Error Indicators** below).

6 Error Indicators

Arccos(x) is undefined for $|x| > 1$. The routine returns zero for any such arguments and the corresponding bit in EMASK is set .TRUE.

7 Auxiliary Routines

None

8 Accuracy

The accuracy is better than 20 parts in 10^7 except for $|x|$ very close to unity, when only 3 or 4 significant figures can be guaranteed.

9 Further Comments

None

10 Keywords

Arccosine, special function

11 Example

The example calculates arccos(x) for 1024 values of x between -1 and 1.

Host program

```
      PROGRAM MAIN
      REAL X(1024), Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
      DO 1 I = 1,1024
      X (I) = FLOAT(I-1)*2.0 / 1023.0 -1.0
    1 CONTINUE
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve data and results from the DAP
C
      CALL DAPREC('XY',X,2048)
```

```
C
C Release the DAP
C
      CALL DAPREL
C
C Write out a sample selection of the data and results for inspection
C
      WRITE (6,2)
    2 FORMAT(6X, 'X', 11X,'Arccos(X)'/)
      DO 3 I = 1,1024,32
    3 WRITE (6,4) X (I), Y(I)
    4 FORMAT(1X,2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,), Y(,)
      LOGICAL EMASK (,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION_S04_ARC_COS
C
C Convert input data
C
      CALL CONVFME(X)
      Y = S04_ARC_COS(X,EMASK)
      IF (ANY(EMASK)) TRACE 1 (EMASK)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

```
      X            Arccos(X)

  -1.0000000        3.141593
   -.9374389        2.785996
   -.8748778        2.635980
   -.8123167        2.518910
   -.7497556        2.418489
   -.6871945        2.328417
   -.6246334        2.245458
   -.5620723        2.167686
   -.4995112        2.093831
   -.4369501        2.023001
   -.3743891        1.954534
   -.3118280        1.887913
   -.2492669        1.822720
   -.1867058        1.758604
   -.1241447        1.695262
   -.6158358e-01    1.632419
    .9775162e-03    1.569818
    .6353867e-01    1.507215
    .1260997        1.444360
    .1886609        1.380998
    .2512219        1.316854
    .3137830        1.251621
    .3763441        1.184949
    .4389052        1.116416
    .5014663        1.045503
    .5640274        .9715414
    .6265885        .8936281
    .6891496        .8104811
    .7517107        .7201442
    .8142718        .6193231
    .8768328        .5015618
    .9393940        .3499371
```

12.2 S04_ARC_SIN

release 1

1 Purpose

S04_ARC_SIN returns the value of the inverse sine function $\arcsin(x)$ for a matrix argument. The result lies in the range $[-\pi/2, \pi/2]$.

2 Specification

REAL MATRIX FUNCTION S04_ARC_SIN (X , EMASK)
REAL X (,)
LOGICAL EMASK (,)

3 Description

Arcsin is approximated using a Tschebyshev polynomial expansion. Since $\arcsin(-x) = -\arcsin(x)$ it is only necessary to consider positive arguments. In the evaluation of arcsin, an expansion is used of the form:

$$p\,(x) = x \sum{}' a_r T_r(t) \quad \text{where} \quad t = 4x^2 - 1$$

where $\sum{}'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum{}'$ is half the first term in $\sum$.

The approximation for different value of the argument x is as follows:

$$\arcsin(x) \simeq p\,(x) \quad \text{where} \quad x \in [0,\ 1/\sqrt{2}]$$

$$\arcsin(x) \simeq \pi/2 - p\,(\sqrt{1-x^2}) \quad \text{where} \quad x \in (1/\sqrt{2}\,, 1]$$

For $|\,x| > 1$ the result is undefined.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 4 section 4, p 79: Dover Publications 1968.

[2] FOX L and PARKER I

Chebyshev Polynomials in Numerical Analysis: Oxford University Press, 1968.

5 Arguments

X – REAL MATRIX

On entry, X contains the points at which the evaluation of arccos is required. All elements of X must be defined on entry. X is unchanged on exit.

EMASK – LOGICAL MATRIX

On exit, EMASK is set .TRUE. at positions corresponding to invalid arguments (see **Error Indicators** below).

6 Error Indicators

Arccos (x) is undefined for $|x| > 1$. The routine returns zero for any such arguments and the corresponding bit in EMASK is set .TRUE.

7 Auxiliary Routines

None

8 Accuracy

The accuracy is better than 20 parts in 10^7 except for $|x|$ very close to unity, when only 3 or 4 significant figures can be guaranteed.

9 Further Comments

None

10 Keywords

Arccosine, special function

11 Example

The example calculates arcsin (x) for 1024 values of x between -1 and 1.

Host program

```
      PROGRAM MAIN
      REAL X(1024) , Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
      DO 1 I = 1,1024
      X(I) = FLOAT (I-1)*2.0/1023.0 - 1.0
 1    CONTINUE
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve data and results from the DAP
C
      CALL DAPREC('XY',X,2048)
```

```
C
C Release the DAP
C
      CALL DAPREL
C
C Write out a sample selection of the data and results for inspection.
C
     WRITE (6,2)
 2    FORMAT(6X,'X',11X, 'Arcsin(X)'/)
      DO 3 I = 1,1024,32
 3    WRITE (6,4) X(I),Y(I)
 4    FORMAT (1X, 2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,),Y(,)
      LOGICAL EMASK(,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION S04_ARC_SIN
C
C Convert input data
C
      CALL CONVFME(X)
      Y = S04_ARC_SIN(X,EMASK)
      IF (ANY(EMASK)) TRACE 1 (EMASK)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

X	Arcsin(X)
-1.0000000	-1.570796
-.9374389	-1.215199
-.8748778	-1.065183
-.8123167	-.9481134
-.7497556	-.8476925
-.6871945	-.7576209
-.6246334	-.6746613
-.5620723	-.5968893
-.4995112	-.5230349
-.4369501	-.4522050
-.3743891	-.3837375
-.3118280	-.3171163
-.2492669	-.2519231
-.1867058	-.1878080
-.1241447	-.1244658
-.6158358e-01	-.6162257e-01
.9775162e-03	.9775162e-03
.6353867e-01	.6358147e-01
.1260997	.1264364
.1886609	.1897984
.2512219	.2539426
.3137830	.3191746
.3763441	.3858470
.4389052	.4543798
.5014663	.5252929
.5640274	.5992549
.6265885	.6771679
.6891496	.7603152
.7517107	.8506517
.8142718	.9514732
.8768328	1.069234
.9393940	1.220859

12.3 S04_ATAN2_M

release 1

1 Purpose

S04_ATAN2_M is a matrix function similar to the standard FORTRAN ATAN2 function. It returns a matrix of values in the range $-\pi$ to π for arc-tangent(matrix-1/matrix-2), in the correct quadrant, and with divide-by-zero errors avoided. If both arguments are zero, zero is returned.

2 Specification

```
REAL MATRIX FUNCTION S04_ATAN2_M(A , B)
REAL A(,) , B(,)
```

3 Description

A logical mask is set up, where each element is defined by the relative magnitudes of the arguments to ATAN2_M. Where the absolute value of an element of matrix A is greater than that of B, the logical mask element is set to .TRUE.; for all other cases the logical mask element is set to .FALSE.

ATAN2_M takes the value:

$$\mathrm{ATAN}\left(\frac{\mathrm{ABS(B)}}{\mathrm{ABS(A)}}\right) \quad \text{where ABS(A)} > \text{ABS(B)} \quad \text{(the logical mask is .TRUE.)}$$

$$\pi/2-\mathrm{ATAN}\left(\frac{\mathrm{ABS(A)}}{\mathrm{ABS(B)}}\right) \quad \text{where ABS(A)} \leq \text{ABS(B)} \quad \text{(the logical mask is .FALSE.)}$$

Thus the built-in ATAN function is always presented with arguments whose values are in the range zero to one, and divide-by-zero errors are avoided, except when the corresponding elements in each argument are zero. After the ATAN operation, the results are corrected to put their values into the correct quadrants, from $-\pi$ to π, according to the signs of the arguments.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, A contains values proportional to the sines of the angles to be returned by the function, and is unchanged on exit.

B – REAL MATRIX

On entry, B contains values proportional to the cosines of the angles to be returned by the function, and is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Over most of the range the results are accurate to within one part in 10^6. Under worst case conditions, where the resultant angle is $\pi/4$, $3\pi/4$, and so on, the error may approach two parts in 10^6.

9 Further Comments

A program interrupt will occur if corresponding elements of A and B are both zero.

10 Keywords

Arc-tangent, inverse tangent

11 Example

In the following example the host routine sets up array ANGLES to contain the radian equivalents of 0, 0.1, 0.2 ... degrees. The DAP routine calculates the sines and cosines of these angles, and then calls S04_ATAN2_M to return the original angles. In this example ANGLES is treated as a long vector.

Host program

```
      PROGRAM MATTESTHOST
      REAL ANGLES(1024)
      COMMON/DAP/ANGLES
C
C Conversion factor from degrees to radians
C
      F=3.14159265/180.0
C
C Initialise data for testing function
C
      DO 1 J=1,1024
 1    ANGLES(J)=FLOAT(J-1)*F*0.1
C
C Connect to DAP module
C
      CALL DAPCON('mattest.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('DAP',ANGLES,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('MATTESTDAP')
C
C Retrieve the results from the DAP
C
```

```
      CALL DAPREC('DAP',ANGLES,1024)
C
C Release the DAP
C
      CALL DAPREL
C
C Write out a sample selection of the data and results for inspection
C
      WRITE(6,2)(J,ANGLES(J),J=1,1024,32)
 2    FORMAT(4('  ',I4,' ',F9.6))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE MATTESTDAP
      REAL*4 SINVALS(,),COSVALS(,),ANGLES(,)
      COMMON/DAP/ANGLES
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL*4 MATRIX FUNCTION S04_ATAN2_M
C
C Convert input data
C
      CALL CONVFME(ANGLES)
C
C Calculate sine and cosine components
C
      SINVALS=SIN(ANGLES)
      COSVALS=COS(ANGLES)
      ANGLES=S04_ATAN2_M(SINVALS,COSVALS)
C
C Convert input results back to host format
C
      CALL CONVMFE(ANGLES)
      RETURN
      END
```

Results

```
   1   .000000    33   .055851    65   .111701    97   .167552
 129   .223402   161   .279253   193   .335103   225   .390954
 257   .446804   289   .502655   321   .558506   353   .614356
 385   .670206   417   .726058   449   .781908   481   .837757
 513   .893608   545   .949459   577  1.005308   609  1.061160
 641  1.117010   673  1.172861   705  1.228711   737  1.284562
 769  1.340412   801  1.396263   833  1.452113   865  1.507964
 897  1.563814   929  1.619666   961  1.675517   993  1.731367
```

12.4 S04_ATAN2_V

release 1

1 Purpose

S04_ATAN2_V is a vector function similar to the standard FORTRAN ATAN2 function. It returns a vector of values in the range $-\pi$ to π for arc-tangent(vector-1/vector-2), in the correct quadrant, and with divide-by-zero errors avoided. If both arguments are zero, zero is returned.

2 Specification

```
REAL VECTOR FUNCTION S04_ATAN2_V (A , B)
REAL A ( ) , B ( )
```

3 Description

A logical mask is set up, where each element is defined by the relative magnitudes of the arguments to S04_ATAN2_V. Where the absolute value of an element of vector A is greater than that of B, the logical mask element is set to .TRUE.; for all other cases the logical mask element is set to .FALSE.

S04_ATAN2_V takes the value:

$$\mathrm{ATAN}\left(\frac{\mathrm{ABS(B)}}{\mathrm{ABS(A)}}\right) \quad \text{where ABS(A) > ABS(B)} \quad \text{(the logical mask is .TRUE.)}$$

$$\pi/2-\mathrm{ATAN}\left(\frac{\mathrm{ABS(A)}}{\mathrm{ABS(B)}}\right) \quad \text{where ABS(A)} \leq \text{ABS(B)} \quad \text{(the logical mask is .FALSE.)}$$

Thus the built-in ATAN function is always presented with arguments whose values are in the range zero to one, and divide-by-zero errors are avoided, except when the corresponding elements in each argument are zero. After the ATAN operation the results are corrected to put their values into the correct quadrants, from $-\pi$ to π, according to the signs of the arguments.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, A contains values proportional to the sines of the angles to be returned by the function, and is unchanged on exit.

B – REAL MATRIX

On entry, B contains values proportional to the cosines of the angles to be returned by the function, and is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Over most of the range the results are accurate to within one part in 10^6. Under worst case conditions, where the resultant angle is $\pi/4$, $3\pi/4$, and so on, the error may approach two parts in 10^6.

9 Further Comments

A program interrupt will occur if corresponding elements of A and B are both zero.

10 Keywords

Arc-tangent, inverse tangent

11 Example

In the following example the host routine sets up array ANGLES to contain the radian equivalents of 0, 6, 12, 18 ... degrees. The DAP routine calculates the sines and cosines of these angles, and then calls S04_ATAN2_V to return the original angles.

Host program

```
      PROGRAM VECTESTHOST
      REAL ANGLES(32)
      COMMON/DAP/ANGLES
C
C Conversion factor from degrees to radians
C
      F=3.14159265/180.0
C
C Initialise data for testing function
C
      DO 1 J=1,32
 1    ANGLES(J)=FLOAT(J-1)*F*6.0
C
C Connect to DAP module
C
      CALL DAPCON('vectest.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('DAP',ANGLES,32)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('VECTESTDAP')
C
C Retrieve the results from the DAP
C
      CALL DAPREC('DAP',ANGLES,32)
```

```
C
C Release the DAP
C
      CALL DAPREL
C
C Write out a sample selection of the data and results for inspection
C
      WRITE(6,2)(J,ANGLES(J),J=1,32)
 2    FORMAT(5('  ',I2,' ',F9.6))
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE VECTESTDAP
      REAL*4 SINVALS(),COSVALS(),ANGLES()
      COMMON/DAP/ANGLES
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL*4 VECTOR FUNCTION S04_ATAN2_V
C
C Convert input data
C
      CALL CONVFVE(ANGLES,32,1)
C
C Calculate sine and cosine components
C
      SINVALS=SIN(ANGLES)
      COSVALS=COS(ANGLES)
      ANGLES=S04_ATAN2_V(SINVALS,COSVALS)
C
C Convert input results back to host format
C
      CALL CONVVFE(ANGLES,32,1)
      RETURN
      END
```

Results

```
  1  .000000   2  .104720   3  .209440   4  .314159   5  .418879
  6  .523599   7  .628318   8  .733039   9  .837757  10  .942478
 11 1.047197  12 1.151917  13 1.256637  14 1.361357  15 1.466076
 16 1.570796  17 1.675517  18 1.780236  19 1.884956  20 1.989675
 21 2.094396  22 2.199116  23 2.303835  24 2.408554  25 2.513275
 26 2.617993  27 2.722714  28 2.827433  29 2.932153  30 3.036873
 31 3.141592  32 -3.036874
```

12.5 S04_COS_INT

release 1

1 Purpose

S04_COS_INT returns the value of the cosine integral $C_i\,(x)$ for a matrix argument.

2 Specification

```
REAL MATRIX FUNCTION S04_COS_INT (X , EMASK)
REAL X (,)
LOGICAL EMASK (,)
```

3 Description

$C_i\,(x)$ is approximated using one of three Tschebyshev expansions. Since $C_i\,(x)$ is imaginary for $x < 0$, only positive arguments are considered. The expansions (and the ranges over which they are valid) are of the form:

$$C_i(x) \simeq \ln(x) + \sum' a_r D_r(t) \quad \text{for} \quad x \in [0,9] \text{ and where } t = 2\left(\frac{x}{9}\right)^2 - 1$$

$$C_i(x) \simeq \ln(x) + \sum' b_r T_r(t) \quad \text{for} \quad x \in (9,16] \text{ and where } t = 2\left(\frac{x-9}{7}\right) - 1$$

$$C_i(x) \simeq f(x) sin(x) - g(x) cos(x) \quad \text{for} \quad x \in (16, \infty)$$

where:

$$f(x) = \sum' c_r T_r(t)$$

$$g(x) = \sum' d_r T_r(t)$$

$$t = 2\left(\frac{16}{x}\right) - 1$$

where $\sum'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum'$ is half the first term in $\sum$.

In the third approximation f and g are asymptotic expansions of the form:

$$f(z) \sim \left(\frac{1}{z}\right)\left\{1 - \left(\frac{2!}{z^2}\right) + \left(\frac{4!}{z^4}\right) - \left(\frac{6!}{z^6}\right) \;\cdots\right\}$$

$$g(z) \sim \left(\frac{1}{z^2}\right)\left\{1 - \left(\frac{3!}{z^2}\right) + \left(\frac{5!}{z^4}\right) - \left(\frac{7!}{z^6}\right) \;\cdots\right\}$$

As $x \rightarrow \infty$, $C_i(x) \rightarrow 0$; this fact is used by the routine for very large arguments.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 4 section 4, p 79: Dover Publications, 1968.

[2] FOX L and PARKER I

Chebyshev Polynomials in Numerical Analysis: Oxford University Press, 1968.

5 Arguments

X – REAL MATRIX

On entry, X contains the points at which the evaluation of C_i is required. All elements of X must be defined on entry, and are unchanged on exit.

EMASK – LOGICAL MATRIX

On exit, EMASK indicates the positions for which the argument was non-positive (see **Error Indicator** below).

6 Error Indicators

$C_i(x)$ is undefined if x is zero, and is imaginary for negative x. In either case the result returned by S04_COS_INT is zero and the corresponding bit in EMASK is set .TRUE.

7 Auxiliary Routines

None

8 Accuracy

In general 6 significant figures of accuracy may be expected in the result. However, close to the zeros of $C_i(x)$ all relative accuracy may be lost. For very large arguments, the result is set to zero as the true value of $C_i(x)$ is less than the possible inaccuracy inherent in 32 bit precision.

9 Further Comments

None

10 Keywords

Cosine integral function, special function

11 Example

The example calculates $C_i(x)$ for 1024 values of x between about 0.005 and 20.

Host program

```
      PROGRAM MAIN
      REAL X(1024),Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
```

```
      DO 1 I = 1,1024
 1    X(I) = FLOAT (I) * 20.0 / 1024.0
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve the results from the DAP
C
      CALL DAPREC('XY',X,2048)
C
C Release the DAP
C
      CALL DAPREL
C
C Write out a sample selection of the data and results for inspection
C
      WRITE (6,2)
 2    FORMAT (6X, 'X', 14X, 'Ci(X) '/)
      DO 3 I = 1,1024, 32
3     WRITE (6,4) X(I),Y(I)
4     FORMAT (1X,2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,), Y(,)
      LOGICAL EMASK (,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION S04_COS_INT
C
C Convert input data
C
      CALL CONVFME(X)
      Y = S04_COS_INT(X, EMASK)
C
C Trace out any components that may be <=0
C
```

```
      IF (ANY(EMASK)) TRACE 1 (EMASK)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

```
      X              Ci(X)

    .1953125e-01  -3.358611
    .6445313       .3590578e-01
    1.269531       .4390500
    1.894531       .4428694
    2.519531       .2795894
    3.144531       .7273293e-01
    3.769531      -.9733582e-01
    4.394531      -.1872826
    5.019531      -.1888952
    5.644531      -.1224203
    6.269531      -.2474308e-01
    6.894531       .6474495e-01
    7.519531       .1165104
    8.144531       .1185551
    8.769531       .7754135e-01
    9.394531       .1383400e-01
    10.01953      -.4708195e-01
    10.64453      -.8390427e-01
    11.26953      -.8623505e-01
    11.89453      -.5696487e-01
    12.51953      -.9857178e-02
    13.14453       .3642845e-01
    13.76953       .6525517e-01
    14.39453       .6775570e-01
    15.01953       .4528236e-01
    15.64453       .7996559e-02
    16.26953      -.2936367e-01
    16.89453      -.5321791e-01
    17.51953      -.5584693e-01
    18.14453      -.3777995e-01
    18.76953      -.7019278e-02
    19.39453       .2435225e-01
```

12.6 S04_MOD_BES_I0

release 1

1 Purpose

S04_MOD_BES_I0 returns the value of the modified Bessel function I0 for a matrix argument.

2 Specification

```
REAL MATRIX FUNCTION S04_MOD_BES_I0(X , EMASK)
REAL X(,)
LOGICAL EMASK(,)
```

3 Description

I0 is approximated using one of three Tschebyshev polynomial expansions. Since the function is even it is only necessary to consider positive arguments. The expansions (and the ranges over which they are valid) are of the form:

$$\mathrm{I0}\,(x) \simeq \exp\,(x) \sum{}' a_i T_i\,(t) \quad \text{for} \quad x \in [0,4] \ \text{and where} \ t = x/2-1$$

$$\mathrm{I0}\,(x) \simeq \exp\,(x) \sum{}' b_i T_i\,(t) \quad \text{for} \quad x \in (4,12] \ \text{and where} \ t = x/4-2$$

$$\mathrm{I0}\,(x) \simeq \frac{\exp\,(x)}{\sqrt{x}} \sum{}' c_i T_i\,(t) \quad \text{for} \quad x \in (12,\infty) \ \text{and where} \ t = 24/x-1$$

where $\sum{}'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum{}'$ is half the first term in $\sum$.

4 References

[1] ABRAMOWITZ M and STEGUN I A
Handbook of Mathematical Functions; chapter 9 , p 374: Dover Publications, 1968.

[2] FOX L and PARKER I
Chebyshev Polynomials in Numerical Analysis: Oxford University Press, 1968.

5 Arguments

X – REAL MATRIX

On entry, X contains the points at which the evaluation of I0 is required. All elements of X must be defined on entry, and are unchanged on exit.

EMASK – LOGICAL MATRIX

On exit, EMASK indicates the positions for which the argument was too large (see **Error Indicator** below).

6 Error Indicators

Since I0 (x) increases rapidly with x, the result could easily overflow even for modest values of x. To prevent this overflow, large values are detected and the corresponding bit in EMASK is set .TRUE. The value returned by the function for such large arguments is that returned by the largest valid argument (that is, an argument of about 174).

7 Auxiliary Routines

None

8 Accuracy

The accuracy depends on the size of the argument. For small arguments (say $|x| < 12$) the error is less than about 20 parts in 10^7, but the error will increase rapidly as $|x|$ increases.

9 Further Comments

None

10 Keywords

Modified Bessel function, special function

11 Example

The example calculates I0 (x) for 1024 values of x between 0 and 20.

Host program

```
      PROGRAM MAIN
      REAL X(1024) , Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
      DO 1 I=1,1024
 1    X(I) = FLOAT(I-1)*20.0/1023.0
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send testdata to the DAP
C
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve the results from the DAP
C
      CALL DAPREC('XY',X,2048)
C
```

```
C Release the DAP
C
      CALL DAPREL
C
      WRITE (6,2)
2     FORMAT(6X, 'X' 14X, 'I0(X)'/)
C
C Write out a sample selection of the data and results for inspection
C
      DO 3 I = 1, 1024 , 32
3     WRITE(6,4) X(I),Y(I)
4     FORMAT(1X, 2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,),Y(,)
      LOGICAL EMASK(,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION S04_MOD_BES_I0
C
C Convert input data
C
      CALL CONVFME(X)
      Y = S04_MOD_BES_I0(X, EMASK)
C
C Trace out a mask to show where arguments were too large
C
      IF (ANY (EMASK))TRACE 1 (EMASK)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

X	I0(X)
.0000000e+00	1.0000000
.6256109	1.100267
1.251222	1.431391
1.876832	2.094550
2.502443	3.295993
3.128055	5.417401
3.753665	9.147502
4.379276	15.72208
5.004888	27.35907
5.630498	48.04684
6.256109	84.97379
6.881721	151.1240
7.507331	269.9973
8.132942	484.2058
8.758554	871.1418
9.384164	1571.584
10.00978	2841.946
10.63539	5149.855
11.26100	9349.078
11.88661	16999.96
12.51222	30957.04
13.13783	56446.73
13.76344	103046.5
14.38905	188319.1
15.01466	344494.9
15.64027	630757.9
16.26588	1155853.
16.89149	2119699.
17.51711	3890039.
18.14272	7143643.
18.76833	.1312658e+08
19.39394	.2413416e+08

12.7 S04_MOD_BES_I1

release 1

1 Purpose

S04_MOD_BES_I1 returns the value of the modified Bessel function I1 for a matrix argument.

2 Specification

```
REAL MATRIX FUNCTION S04_MOD_BES_I1(X , EMASK)
REAL X(,)
LOGICAL EMASK(,)
```

3 Description

I1 is approximated using 3 Tschebyshev polynomial expansions. Since $I1(-x) = -I1(x)$ it is only necessary to consider positive arguments. The expansions (and the ranges over which they are valid) are of the form:

$$I1(x) \simeq x \sum{}' a_i T_i(t) \quad \text{for} \quad x \in [0,4] \text{ and where } t = x/2-1$$

$$I1(x) \simeq \exp(x) \sum{}' b_i T_i(t) \quad \text{for} \quad x \in (4,12] \text{ and where } t = x/4-2$$

$$I1(x) \simeq \frac{\exp(x)}{\sqrt{x}} \sum{}' c_i T_i(t) \quad \text{for} \quad x \in (12,\infty) \text{ and where } t = 24/x-1$$

where $\sum{}'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum{}'$ is half the first term in $\sum$.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 9, p 374: Dover Publications

[2] FOX L and PARKER I

Chebyshev Polynomials in Numerical Analysis: Oxford University Press, 1968

5 Arguments

X – REAL MATRIX

On entry X contains the points at which the evaluation of I1 is required. All elements of X must be defined on entry. X is unchanged on exit.

EMASK – LOGICAL MATRIX

On exit EMASK indicates the positions for which the argument was too large (see **Error Indicators** below).

6 Error Indicators

Since I1 (x) increases rapidly with x, the result could easily overflow even for modest values of x. To prevent this, large values are detected and the corresponding bit in EMASK is set .TRUE. The value returned by the function for such large arguments is that returned by the largest valid argument (that is, an argument of about 174).

7 Auxiliary Routines

None

8 Accuracy

The accuracy depends on the size of the argument. For small arguments (say $|x| < 12$) the error is less than about 20 parts in 10^7, but the error will increase rapidly as $|x|$ increases.

9 Further Comments

None

10 Keywords

Modified Bessel function, special function

11 Example

The example calculates I1 (x) for 1024 values of x between 0 and 20.

Host program

```
      PROGRAM MAIN
      REAL X(1024),Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
      DO 1 I = 1,1024
 1    X(I) = FLOAT(I1)*20.0/1023.0
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send testdata to the DAP
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve the results from the DAP
C
      CALL DAPREC('XY',X,2048)
```

```
C
C Release the DAP
C
      CALL DAPREL
C
      WRITE (6,2)
 2    FORMAT(6X,'X',14X,'I1(X)'/)
C
C Write out a sample selection of the data and results for inspection.
C
      DO 3 I = 1,1024,32
 3    WRITE (6,4) X (I) , Y(I)
 4    FORMAT (1X,2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,),Y(,)
      LOGICAL EMASK(,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION S04_MOD_BES_I1
C
C Convert input data
C
      CALL CONVFME(X)
      Y=S04_MOD_BES_I1(X,EMASK)
C
C Trace out a mask to show where arguments were too large
C
      IF (ANY(EMASK))TRACE 1 (EMASK)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

X	I1(X)
.0000000e+00	.0000000e+00
.6256109	.3283606
1.251222	.7562914
1.876832	1.416910
2.502443	2.522305
3.128055	4.436992
3.753665	7.805889
4.379276	13.78311
5.004888	24.44524
5.630498	43.54034
6.256109	77.84995
6.881721	139.6668
7.507331	251.3122
8.132942	453.3796
8.758554	819.7910
9.384164	1485.328
10.00978	2696.016
10.63539	4901.422
11.26100	8923.789
11.88661	16268.36
12.51222	29692.97
13.13783	54254.05
13.76344	99229.38
14.38905	181652.6
15.01466	332817.7
15.64027	610248.1
16.26588	1119740.
16.89149	2055966.
17.51711	3777319.
18.14272	6943891.
18.76833	.1277195e+08
19.39394	.2350347e+08

12.8 S04_SIN_INT

release 1

1 Purpose

S04_SIN_INT returns $Si(x) = \int_0^x \frac{\sin(u)}{u} du$ for a matrix argument.

2 Specification

```
REAL MATRIX FUNCTION S04_SIN_INT (X)
REAL X (,)
```

3 Description

$S_i(x)$ is approximated using one of three Tschebyshev polynomial expansions.
$S_i(-x) = S_i(x)$, so it is only necessary to consider positive arguments. The expansions (and the ranges over which they are valid) are of the form:

$$S_i(x) \simeq x \sum{}' a_r T_r(t) \quad \text{for} \quad x \in [0,9] \text{ and where } t = 2\left(\frac{x}{9}\right)^2 - 1$$

$$S_i(x) \simeq x \sum{}' b_r T_r(t) \quad \text{for} \quad x \in (9,16] \text{ and where } t = 2\left(\frac{x-9}{7}\right) - 1$$

$$S_i(x) \simeq \pi/2 - f(x)\cos(x) - g(x)\sin(x) \quad \text{for} \quad x \in (16, \infty)$$

where:

$$f(x) = \sum{}' c_r T_r(t)$$

$$g(x) = \sum{}' d_r T_r(t)$$

$$t = 2\left(\frac{16}{x}\right) - 1$$

$\sum{}'$ is a series equal, term for term, to $\sum$, except that the first term in $\sum{}'$ is half the first term in $\sum$

In the third approximation f and g are asymptotic expansions of the form:

$$f(z) \sim \left(\frac{1}{z}\right)\left\{1 - \left(\frac{2!}{z^2}\right) + \left(\frac{4!}{z^4}\right) - \left(\frac{6!}{z^6}\right) \ \cdots \right\}$$

$$g(z) \sim \left(\frac{1}{z^2}\right)\left\{1 - \left(\frac{3!}{z^2}\right) + \left(\frac{5!}{z^4}\right) - \left(\frac{7!}{z^6}\right) \ \cdots \right\}$$

As $x \to \pm\infty$, $S_i(x) \to \pm\pi/2$; this fact is used by the routine for very large arguments.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 5 section 2, p 231: Dover Publications, 1968.

[2] FOX L and PARKER I

Chebyshev Polynomials in Numerical Analysis: Oxford University Press; 1968.

5 Arguments

X – REAL MATRIX

On entry, X contains the points at which the evaluation of S_i is required. All elements of X must be defined on entry, and are unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

The maximum error should be less than about 20 parts in 10^7.

9 Further Comments

None

10 Keywords

Sine integral function, special function

11 Example

The example calculates S_i for 1024 values of x between −10 and 10.

Host program

```
      PROGRAM MAIN
      REAL X(1024) , Y(1024)
      COMMON /XY/X,Y
C
C Initialise data for testing function
C
      DO 1 I = 1, 1024
 1    X(I) = FLOAT (I-1)*20.0 / 1023.0 - 10.0
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
```

```
C
C Send testdata to the DAP
C
      CALL DAPSEN('XY',X,1024)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Retrieve the results from the DAP
C
      CALL DAPREC('XY',X,2048)
C
C Release the DAP
C
      CALL DAPREL
C
      WRITE (6,2)
 2    FORMAT(6X, 'X', 14X, 'Si(X)'/)
C
C Write out a sample selection of the data and results for inspection
C
      DO 3 I = 1,1024,32
 3    WRITE (6,4) X(I), Y(I)
 4    FORMAT(1X, 2G15.7)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      REAL X(,) , Y(,)
      COMMON /XY/X,Y
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL REAL MATRIX FUNCTION S04_SIN_INT
C
C Convert input data
C
      CALL CONVFME(X)
      Y= S04_SIN_INT (X)
C
C Convert input data and results back to host format
C
      CALL CONVMFE(X)
      CALL CONVMFE(Y)
      RETURN
      END
```

Results

X	Si(X)
-10.000000	-1.658348
-9.374389	-1.674626
-8.748778	-1.650258
-8.123167	-1.589128
-7.497556	-1.510376
-6.871944	-1.443393
-6.246334	-1.418261
-5.620723	-1.454368
-4.995112	-1.550871
-4.369501	-1.682421
-3.743890	-1.802158
-3.118279	-1.851851
-2.492668	-1.776752
-1.867058	-1.541133
-1.241446	-1.139938
-.6158361	-.6030076
.9775162e-02	.9775121e-02
.6353865	.6213064
1.260997	1.154774
1.886608	1.551066
2.512219	1.781414
3.137830	1.851934
3.763441	1.799163
4.389051	1.678203
5.014663	1.547130
5.640274	1.452258
6.265884	1.418175
6.891495	1.444993
7.517107	1.512826
8.142717	1.591439
8.768328	1.651638
9.393940	1.674711

12.9 S15_ERF

release 1

1 Purpose

S15_ERF returns the value of the error function.

2 Specification

```
REAL*8 MATRIX FUNCTION S15_ERF(X)
REAL*8 X(,)
```

3 Description

The function is calculated by one of three algorithms. The algorithms used (and the ranges over which they are valid) are:

$$|\mathrm{erf}(x)| = |x|\,\mathrm{T}_1(\mathrm{T}) \quad \text{for} \quad |x| \in [0,2] \text{ and where } \mathrm{T} = \frac{x^2}{2} - 1$$

$$|\mathrm{erf}(x)| = 1 - \frac{\exp(-x^2)}{|x|\sqrt{\pi}}\,\mathrm{T}_2(\mathrm{T}) \quad \text{for} \quad |x| \in (2,\mathrm{XHIGH}) \text{ and where } T = \frac{x-7}{x+3}$$

$$|\mathrm{erf}(x)| = 1 \quad \text{for} \quad |x| \in [\mathrm{XHIGH},\infty]$$

where XHIGH is the value above which $\mathrm{erf}(x) = 1$, to the machine's accuracy; XHIGH is machine-dependent, and is 6.25 for the DAP

The sign of erf$x$) is the same as that of x; $\mathrm{T}_1(\mathrm{T})$ and $\mathrm{T}_2(\mathrm{T})$ are Tschebychev polynomial expansions. They are evaluated using recursive descent by the function 'ZTSCHEB', which has as parameters the dimension and array of coefficients for the expansion. The argument 'T' is passed in the named common block 'CTSCHEBARG'.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 7 section 1, p 297: Dover Publications, 1968.

5 Arguments

X – REAL*8 MATRIX

On entry, X contains the points at which the function is to be evaluated. All elements of X must be assigned on entry; X is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routine ZTSCHEB.

8 Accuracy

The DAP works to a precision of about 17 significant figures in REAL*8 arithmetic. S15_ERF was checked against S15_ERFC according to the relation:

$$\mathrm{erf}\,(x) + \mathrm{erfc}\,(x) = 1$$

The worst error was 7 E-16, and the median error was about 2 E-16.

9 Further Comments

The routine uses the common block 'CTSCHEBARG' to pass a parameter to the function 'ZTSCHEB', so you must not use a block of that name.

10 Keywords

Error function, special function

11 Example

The following example program reads and prints a caption and then reads pairs of numbers from the data stream. The program assumes that the first number of each pair indicates whether the second number in the pair is a valid argument of the function. Reading of the pairs of numbers continues until the first number in a pair is negative.

The program packs the arguments into the first column of a 32 by 32 array, X, which is passed by the named common block COM1 to the DAP entry subroutine DAPSUB. The subroutine converts the values into DAP storage mode, then calls S15_ERF. The result is assigned to matrix Y, which is also in common block COM1. Both matrices are converted back into host storage mode and the results printed.

Host program

```
      PROGRAM MAIN

      INTEGER INUM(32,32)
      CHARACTER*40 TITLE
      COMMON /COM1/X(32,32),Y(32,32)
      DOUBLE PRECISION X,Y
C
C Initialise X to avoid 'UNASSIGNED VARIABLE'
C
      DO 2 J = 1,32
      DO 1 I = 1,32
      X(I,J) = 0.0
 1    CONTINUE
 2    CONTINUE
C
      READ (*,5) TITLE
      WRITE (*,6) TITLE
      WRITE (*,7)
```

```
C
C Read data
C

      J=0
 3    J=J+1
      READ (*,8) INUM(J,1), X(J,1)
      IF (INUM(J,1).GE.0)GOTO3
C
C Connect to DAP module
C
      CALL DAPCON('dapsub.dd')
C
C Send test data to DAP
C
      CALL DAPSEN('COM1',X,2048)
C
C Call DAP routine
C
      CALL DAPENT('DAPSUB')
C
C Receive test data and results from DAP
C
      CALL DAPREC('COM1',X,4096)
C
C Release the DAP
C
      CALL DAPREL
C
C Write out results
C
      J = J - 1
      DO 4 I=1,J
 4    WRITE (*,9)  X(I,1), Y(I,1), INUM(I,1)
      STOP
 5    FORMAT (A)
 6    FORMAT (4(1X/), 1H , A, 8H RESULTS/1X)
 7    FORMAT (18X, 'X', 25X, 'Y', 13X, 'INUM')
 8    FORMAT (I5, F20.5)
 9    FORMAT (4X, 1PD20.3, 1X, 1PD20.3, 14X, I2)
      END
```

DAP program

```
      ENTRY SUBROUTINE DAPSUB
C
C Note the use of the external statement for this function
C
     EXTERNAL REAL*8 MATRIX FUNCTION S15_ERF
      COMMON /COM1/ X(,),Y(,)
      REAL*8 X,Y
```

```
C
C Convert input data
C
      CALL CONVFMD(X)
C
      Y(,) = S15_ERF(X)
C
C Convert input data and results back to host mode
C
      CALL CONVMFD(X)
      CALL CONVMFD(Y)
      RETURN
      END
```

Data

```
S15ERF EXAMPLE PROGRAM DATA
    1            -6.0
    2            -4.5
    3            -1.0
    4             1.0
    5             4.5
    6             6.0
   -1             0.0
```

Results

```
 S15ERF EXAMPLE PROGRAM DATA                RESULTS

                   X                        Y             INUM
              -6.000e+00              -1.000e+00            1
              -4.500e+00              -1.000e+00            2
              -1.000e+00              -8.427e-01            3
               1.000e+00               8.427e-01            4
               4.500e+00               1.000e+00            5
               6.000e+00               1.000e+00            6
```

12.10 S15_ERFC

release 1

1 Purpose

S15_ERFC returns the value of the complement of the error function.

2 Specification

REAL*8 MATRIX FUNCTION S15_ERFC(X)
REAL*8 X(,)

3 Description

S15_ERFC returns the complement of the error fucntion S15_ERC. S15_ERFC is calculated by one of four algorithms. The algorithms used (and the ranges over which they are valid) are:

$$\mathrm{erfc}(x) = 2.0 \text{ (to machine accuracy)} \quad \text{for } x \in (-\infty, \mathrm{XLOW})$$

$$\mathrm{erfc}(x) = 2.0 - \exp(-x^2)\ \mathrm{POLY}(\mathrm{T}) \quad \text{for } x \in [\mathrm{XLOW}, 0)$$

$$\mathrm{erfc}(x) = \exp(-x^2)\ \mathrm{POLY}(\mathrm{T}) \quad \text{for } x \in [0, \mathrm{XHIGH})$$

$$\mathrm{erfc}(x) = 0.0 \text{ (to machine accuracy)} \quad \text{for } x \in [\mathrm{XHIGH}, \infty)$$

where:

XLOW and XHIGH are values that are machine-dependent; for the DAP they are –6.25 and 13.0 respectively

POLY(T) is a Tschebychev polynomial function of T, where:

$$\mathrm{T} = \frac{|x| - 3.75}{|x| + 3.75}$$

and is calculated by conversion to an ordinary polynomial, which is then evaluated by Horner's method.

4 References

[1] ABRAMOWITZ M and STEGUN I A

Handbook of Mathematical Functions; chapter 7 section 1, p 297: Dover Publications, 1968.

5 Arguments

X – REAL*8 MATRIX

On entry, X contains the points at which the function is to be evaluated. All elements of X must be assigned on entry; X is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

If E and D are the relative errors in result and argument respectively, they are in principle related by:

$$|E| = \left|\frac{2x\exp(-x^2)}{\sqrt{\pi}\ \mathrm{erfc}\,(x)}\right| D$$

You should note that near $x = 0$ the amplification factor behaves as $\frac{2x}{\sqrt{\pi}}$, hence the accuracy is also largely determined by machine precision.

For large negative x, where the factor is $x\frac{\exp(-x^2)}{\sqrt{\pi}}$, accuracy is mainly limited by machine precision.

For large positive x ,the factor behaves like $2x^2$ and hence to a certain extent relative accuracy is unavoidably lost. However the absolute error in the result, E, is given by:

$$|E| = \left|\frac{2x\exp(-x^2)}{\sqrt{\pi}}\right| D$$

so absolute accuracy can be guaranteed for all x.

9 Further Comments

None

10 Keywords

Complementary error function, special function.

11 Example

The following example program reads and prints a caption and then reads pairs of numbers from the data stream. The program assumes that the first number of each pair indicates whether the second number in the pair is a valid argument of the function. Reading of the pairs of numbers continues until the first number in a pair is negative.

The program packs the arguments into the first column of a 32 by 32 array, X, which is passed by the named common block COM1 to the DAP entry subroutine DAPSUB. The subroutine converts the values into DAP storage mode, then calls S15_ERFC. The result is assigned to matrix Y, which is also in common block COM1. Both matrices are converted back into host storage mode and the results printed.

Host program

```
      PROGRAM MAIN
C
C S15_ERFC example program
C
      INTEGER IFAIL(32,32)
      CHARACTER*40 TITLE
      COMMON /COM1/X(32,32),Y(32,32)
      DOUBLE PRECISION X,Y
```

```
C
C Initialise X to avoid
C 'UNASSIGNED VARIABLE'
C
      DO 2 J = 1,32
      DO 1 I = 1,32
      X(I,J) = 0.0
 1    CONTINUE
 2    CONTINUE
C
      READ (*,5) TITLE
      WRITE (*,6) TITLE
      WRITE (*,7)
C
C Read data
C

      J=0
3     J=J+1
      READ (*,8) IFAIL(J,1), X(J,1)
      IF (IFAIL(J,1).GE.0)GOTO3
C
C Connect to DAP module
C
      CALL DAPCON('dapsub.dd')
C
C Send test data to DAP
C
      CALL DAPSEN('COM1',X,2048)
C
C Call DAP routine
C
      CALL DAPENT('DAPSUB')
C
C Receive test data and results from DAP
C
      CALL DAPREC('COM1',X,4096)
C
C Release the DAP
C
     CALL DAPREL
```

```
C
C Write out results
C
      J = J - 1
      DO 4 I=1,J
 4    WRITE (*,9)  X(I,1), Y(I,1), IFAIL(I,1)
 5    FORMAT (A)
 6    FORMAT (4(1X/), 1H , A, 8H RESULTS/1X)
 7    FORMAT (18X, 'X', 25X, 'Y',13X, 'INUM')
 8    FORMAT (I5, F20.5)
 9    FORMAT (4X, 1PD20.3, 1X, 1PD20.3, 14X, I2)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE DAPSUB
C
C Note the use of the external statement for this function
C
      EXTERNAL REAL*8 MATRIX FUNCTION S15_ERFC
      COMMON /COM1/ X(,),Y(,)
      REAL*8 X,Y
C
C Convert input data
C
      CALL CONVFMD(X)
C
      Y(,) = S15_ERFC(X)
C
C Convert input data and results back to host mode
C
      CALL CONVMFD(X)
      CALL CONVMFD(Y)
      RETURN
      END
```

Data

```
S15ERFC EXAMPLE PROGRAM DATA
    1           -10.0
    2            -1.0
    3             0.0
    4             1.0
    5            15.0
   -1             0.0
```

Results

```
S15ERFC EXAMPLE PROGRAM DATA            RESULTS

                 X                        Y              INUM
            -1.000e+01              2.000e+00              1
            -1.000e+00              1.843e+00              2
             0.000e+00              1.000e+00              3
             1.000e+00              1.573e-01              4
             1.500e+01              0.000e+00              5
```

Chapter 13

X01 – Mathematical constants

Contents:

13.1 X01_PI

release 1

1 Purpose

X01_PI provides the value of pi for any of the real precision lengths available on the DAP.

2 Specification

```
SUBROUTINE X01_PI(PI , LEN)
REAL* <LEN> PI
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

PI – REAL* <LEN>

On exit, PI contains the value of π for reals of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of PI (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

This routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results are to machine accuracy for the precision required.

9 Further Comments

None

10 Keywords

Machine constants, pi

11 Example

The following FORTRAN-PLUS fragment traces out the REAL*4 value for π.

```
      ENTRY SUBROUTINE ENT
      REAL*4 PI
      CALL X01_PI(PI,4)
      TRACE 1 (PI)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Real Scalar Local Variable PI in 32 bits - on Stack at 0.09

 3.1415930E+00

End of Report
```

Chapter 14

X02 – Machine constants

Contents:

14.1 X02_EPSILON

release 1

1 Purpose

X02_EPSILON provides the smallest positive real (EPS) such that 1.0 + EPS differs from 1.0, for any of the real precision lengths available on the DAP.

2 Specification

```
SUBROUTINE X02_EPSILON (EPSILON, LEN)
REAL* <LEN> EPSILON
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

EPSILON – REAL* <LEN>

On exit, EPSILON contains the value of EPS for reals of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of EPSILON (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results are to machine accuracy for the precision required.

9 Further Comments

None

10 Keywords

Machine constants, machine precision

11 Example

The following FORTRAN-PLUS fragment traces out the REAL*4 value of ϵ.

```
      ENTRY SUBROUTINE ENT
      REAL*4 EPS
      CALL X02_EPSILON(EPS,4)
      TRACE 1 (EPS)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Real Scalar Local Variable EPS in 32 bits - on Stack at 0.09

 9.5367432E-07

End of Report
```

14.2 X02_MAXDEC

release 1

1 Purpose

X02_MAXDEC provides a value for MAXDEC for the range of reals of different precision available on the DAP; MAXDEC is the maximum number of decimal digits which can be accurately represented over the whole range of floating point numbers.

2 Specification

```
SUBROUTINE X02_MAXDEC(M , LEN)
INTEGER M , LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

M – INTEGER

On exit, M contains the value of MAXDEC for reals of length LEN bytes.

LEN – INTEGER

On entry LEN must contain the length in bytes of the reals for which MAXDEC is required (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. Unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Whilst the results given are accurate for any particular real number, precision may be lost after a sequence of arithmetic operations.

9 Further Comments

None

10 Keywords

Machine constants, real precision

11 Example

The following FORTRAN-PLUS fragment traces out the maximum number of decimal digits which can be accurately represented over the whole range of REAL*4 precision floating point numbers.

```
ENTRY SUBROUTINE ENT
INTEGER MAXD
CALL X02_MAXDEC(MAXD,4)
TRACE 1 (MAXD)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Integer Scalar Local Variable MAXD in 32 bits - on Stack at 0.09

          6

End of Report
```

14.3 X02_MAXINT

release 1

1 Purpose

X02_MAXINT provides a value for MAXINT for the range of integers of different precision available on the DAP; MAXINT is the largest integer such that MAXINT and -MAXINT can both be represented exactly.

2 Specification

```
SUBROUTINE X02_MAXINT(M , LEN)
INTEGER* <LEN> M
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

M – INTEGER* <LEN>

On exit, M contains the value of MAXINT for integers of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of M (in the range 1 to 8). If LEN is outside the range 1 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results returned are to machine accuracy for the precision required.

9 Further Comments

None

10 Keywords

Machine constants, maximum integer

11 Example

The following FORTRAN-PLUS fragment traces out the value of MAXINT for INTEGER*4 precision.

```
ENTRY SUBROUTINE ENT
INTEGER MAXI
CALL X02_MAXINT(MAXI,4)
TRACE 1 (MAXI)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Integer Scalar Local Variable MAXI in 32 bits - on Stack at 0.09

 2147483647

End of Report
```

14.4 X02_MAXPW2

release 1

1 Purpose

X02_MAXPW2 provides a value for MAXPW2 for the range of reals of different precision available on the DAP; MAXPW2 is the largest integer power to which 2.0 may be raised without overflow.

2 Specification

SUBROUTINE X02_MAXPW2(M)
INTEGER* <2–4> M

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

M – INTEGER* <2–4>

On exit, M contains the value of MAXPW2 for reals of any length

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

The accuracy does not depend on the precision used.

9 Further Comments

None

10 Keywords

Machine constants, maximum power of 2

11 Example

The following FORTRAN-PLUS fragment traces out the largest integer power to which 2.0 may be raised without overflow for any real precision length.

```
ENTRY SUBROUTINE ENT
INTEGER MAXPW2
CALL X02_MAXPW2(MAXPW2)
TRACE 1 (MAXPW2)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Integer Scalar Local Variable MAXPW2 in 32 bits - on Stack at 0.09

        251

End of Report
```

14.5 X02_MINPW2

release 1

1 Purpose

X02_MINPW2 provides a value for MINPW2 for the range of reals of different precision available on the DAP; MINPW2 is the largest negative integer power to which 2.0 may be raised without underflow.

2 Specification

SUBROUTINE X02_MINPW2(M)
INTEGER* <2–4> M

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

M – INTEGER* <2–4>

On exit, M contains the value of MINPW2 for reals of any length.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

The accuracy does not depend on the precision used.

9 Further Comments

None

10 Keywords

Machine constants, maximum negative power of 2

11 Example

The following FORTRAN-PLUS fragment traces out the largest negative integer power to which 2.0 may be raised without underflow for any real precision length.

```
ENTRY SUBROUTINE ENT
INTEGER MINPW2
CALL X02_MINPW2(MINPW2)
TRACE 1 (MINPW2)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Integer Scalar Local Variable MINPW2 in 32 bits - on Stack at 0.09

251

End of Report
```

14.6 X02_RMAX

release 1

1 Purpose

X02_RMAX provides the largest real (RMAX) such that RMAX and -RMAX can both be represented exactly, for the range of reals of different precision available on the DAP.

2 Specification

```
SUBROUTINE X02_RMAX(R , LEN)
REAL* <LEN> R
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

R – REAL* <LEN>

On exit, R contains the value of RMAX for reals of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of R (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results returned are as accurate as possible for the precision required.

9 Further Comments

None

10 Keywords

Machine constants, maximum real value.

11 Example

The following FORTRAN-PLUS fragment traces out the value of RMAX for REAL*4 precision.

```
ENTRY SUBROUTINE ENT
REAL*4 RMAX
CALL X02_RMAX(RMAX,4)
TRACE 1 (RMAX)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Real Scalar Local Variable RMAX in 32 bits - on Stack at 0.09

 7.2370051E+75

End of Report
```

14.7 X02_RMIN

release 1

1 Purpose

X02_RMIN provides the smallest real (RMIN) such that RMIN and –RMIN can both be represented exactly, for the range of reals of different precision available on the DAP.

2 Specification

```
SUBROUTINE X02_RMIN (R , LEN)
REAL* <LEN> R
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

R – REAL* <LEN>

On exit, R contains the value of RMIN for reals of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of R (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results returned are as accurate as possible for the precision required.

9 Further Comments

None

10 Keywords

Machine constants, minimum real value.

11 Example

The following FORTRAN-PLUS fragment traces out the value of RMIN for REAL*4 precision.

```
      ENTRY SUBROUTINE ENT
      REAL*4 RMIN
      CALL X02 RMIN(RMIN,4)
      TRACE 1 (RMIN)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Real Scalar Local Variable RMIN in 32 bits - on Stack at 0.09

 5.3976053E-79

End of Report
```

14.8 X02_TOL

release 1

1 Purpose

X02_TOL provides the value of TOL (= RMIN/EPSILON) for the range of reals of different precision available on the DAP.

2 Specification

```
SUBROUTINE X02_TOL (R , LEN)
REAL* <LEN> R
INTEGER LEN
```

3 Description

The relevant value is picked out from a table of values.

4 References

None

5 Arguments

R – REAL* <LEN>

On exit, R contains the value of TOL for reals of length LEN bytes.

LEN – INTEGER

On entry, LEN must contain the length in bytes of R (in the range 3 to 8). If LEN is outside the range 3 to 8 the results are unpredictable. LEN is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine references the General Support library routine Z_X01_X02_AUX.

8 Accuracy

The results returned are as accurate as possible for the precision required.

9 Further Comments

None

10 Keywords

Machine constants

11 Example

The following FORTRAN-PLUS fragment traces out the value of TOL for REAL*4 precision.

```
ENTRY SUBROUTINE ENT
REAL*4 TOL
CALL X02_TOL(TOL,4)
TRACE 1(TOL)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 4

Real Scalar Local Variable TOL in 32 bits - on Stack at 0.09

 5.6597994E-73

End of Report
```

Chapter 15

X05 – Other utilities

Contents:

15.1 X05_ALT_LV

release 1

1 Purpose

X05_ALT_LV produces a long vector of alternating groups of N false values followed by N true values and so on until all components of the vector have a value. If the value of N lies outside the range 1 to 1024 all components will have the value .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_ALT_LV(N)
INTEGER N
```

3 Description

The required pattern is set up by first producing a long vector containing the values 0 to 1023 in long vector order. The vector is divided by N and the required pattern supplied by the least significant bit plane of the resulting vector.

4 References

None

5 Arguments

N – INTEGER

On entry, N specifies the number of false and true values to be repeated alternately. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the DAP library routine X05_LONG_INDEX.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

None

11 Example

This FORTRAN-PLUS fragment demonstrates the use of the function X05_ALT_LV to initialise alternate groups of five elements of the long vector X with different values.

```
SUBROUTINE TLVA

REAL X(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_ALT_LV

LM=X05_ALT_LV(5)
X=0.0
X(LM)=1.0

RETURN
END
```

15.2 X05_CRINKLE

release 1

1 Purpose

X05_CRINKLE effects a transformation in data storage format for vertical mode data occupying an array of matrices – from 'sliced' to 'crinkled' storage.

2 Specification

SUBROUTINE X05_CRINKLE (S, L, NR, NC, IFAIL)
<*any type, any length*> S (, , NR, NC)
INTEGER L, NR, NC, IFAIL

3 Description

The data is conceptually considered to occupy an array C of components of size 32 NR by 32 NC. (NR or NC are positive integers, not excluding 1). The storage area, S, is an NR by NC array of matrices. In the 'sliced' format:

$$S(i_r, i_c, j_r, j_c) = C(i_r+32(j_r-1), i_c+32(j_c-1))$$

that is, each value of j_r selects a contiguous group of 32 rows of C, and so on.

In the 'crinkled' format:

$$S(i_r, i_c, j_r, j_c) = C(j_r+\mathrm{NR}\, i_{r-1}, j_c+\mathrm{NC}(i_c-1))$$

that is, each value of i_r selects a contiguous group of NR rows of C, and so on.

In the 'sliced' format the conceptual array is divided into subarrays of size 32 by 32. In the 'crinkled' format the conceptual array is divided into subarrays of size NR by NC.

To carry out the transformation, first a mapping transformation is done on East – West vertical sections of the data area. Each section is regarded as an array of 32 NC data items; each item is of length L by NR (vertical) bits. The transformation reverses the mapping order so that succesive horizontal sets of NC data items are rethreaded vertically.

Then a similar transformation is done on NC separate groups of North – South vertical sections of the data area. Each section of each group is regarded as an array of 32 NR data items; each item is of length L (vertical) bits. The transformation reverses the mapping order so that successive horizontal sets of NR data items are rethreaded vertically.

4 References

None

5 Arguments

S – <*any type, any length*> MATRIX array of dimension (, , NR, NC)

On entry, S contains the sliced data to be reformatted. On exit, S contains the data in crinkled form.

L – INTEGER

On entry, L specifies the length in bits of the components of S; L is unchanged on exit.

NR – INTEGER

On entry, NR specifies the first unconstrained dimension of S; NR is unchanged on exit.

5 Arguments – continued

NC – INTEGER

On entry, NC specifies the second unconstrained dimension of S; NC is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	either NR or NC was less than 1
IFAIL = 2	L was less than 1

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Crinkled data storage, data formatting, data movement, sliced data storage

11 Example

This FORTRAN-PLUS fragment shows how the routine can be used in an entry subroutine to convert a matrix set from sliced to crinkled form.

```
      ENTRY SUBROUTINE ENT
      REAL A(,,2,2)
      COMMON /A/A
      DO 10 I=1,2
      DO 10 J=1,2
      CALL CONVFME(A(,,I,J))
   10 CONTINUE
      CALL X05_CRINKLE(A,32,2,2,IFAIL)
      IF (IFAIL.NE.0) RETURN
C     DAP processing
      RETURN
      END
```

15.3 X05_EAST_BOUNDARY

release 1

1 Purpose

X05_EAST_BOUNDARY returns a logical matrix containing at most one .TRUE. element in each row, corresponding to the last .TRUE. (if any) in each row of the logical matrix parameter. That is, the subroutine is equivalent to the FORTRAN-PLUS code:

```
      KM = .FALSE.
      DO 10 I = 1, 32
      IF (.NOT. ANY(LM(I,))) GOTO 10
      KM(I,) = REV(FRST(REV(LM(I,))))
   10 CONTINUE
```

2 Specification

```
LOGICAL MATRIX FUNCTION X05_EAST_BOUNDARY(LM)
LOGICAL LM(,)
```

3 Description

The DAP store plane (logical matrix LM) passed to the routine is treated as a set of 32 logical vectors, arranged so that each vector occupies a complete row. Each of these vectors is dealt with independently, but in parallel.

To each vector is ripple-added a row of all true bits; the easternmost bit of the vector is treated as least significant. The addition is thrown away; the row of carry bits from the addition, and a shifted-west version of the row of carries, are XORed to give a vector with only one true element: the easternmost .TRUE. element in each input vector. The 32 resultant vectors, produced in parallel, form the required east boundary matrix.

4 References

None

5 Arguments

LM – LOGICAL MATRIX

On entry, LM is the logical matrix whose east boundary is required. LM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Boundary

11 Example

This FORTRAN-PLUS fragment takes a 'black and white' logical matrix (a chess board pattern) as input and returns the east boundary.

```
ENTRY SUBROUTINE ENT
LOGICAL LM(,),KM(,)
EXTERNAL LOGICAL MATRIX FUNCTION X05_EAST_BOUNDARY
LM=ALTR(1).LEQ.ALTC(1)
KM=X05_EAST_BOUNDARY(LM)
TRACE 1 (KM)
RETURN
END
```

The result in this case is simply LM .AND. COLS(31,32)

15.4 X05_E_MAX_PC

release 1

1 Purpose

X05_E_MAX_PC returns a logical matrix marking the maximum value(s) in each row of the real matrix argument. The i^{th} row of the argument contains one or more elements whose value is the maximum value for the row. The corresponding element(s) of the i^{th} row of the logical matrix are set to .TRUE. to mark that maximum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_E_MAX_PC(RM)
REAL RM(,)
```

3 Description

In each row of the argument which contains at least one positive number the position(s) of the maximum positive number is found, and the corresponding output logical mask element(s) set to .TRUE. If a row of the argument contains only negative numbers, the position(s) of the elements with smallest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose row-wise maximum positions are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each row of the matrix processed in the following FORTRAN-PLUS fragment the maximum value(s) in that row are replaced by the value 0.0.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_E_MAX_PC

LM = X05_E_MAX_PC(RM)
RM(LM) = 0.0
RETURN
END
```

15.5 X05_E_MAX_PR

release 1

1 Purpose

X05_E_MAX_PR returns a logical matrix marking the maximum value(s) in each column of the real matrix argument. The i^{th} column of the argument contains one or more elements whose value is the maximum value for the column. The corresponding element(s) of the i^{th} column of the logical matrix are set to .TRUE. to mark that maximum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_E_MAX_PR(RM)
REAL RM(,)
```

3 Description

In each column of the argument which contains at least one positive number the position(s) of the maximum positive number is found, and the corresponding output logical mask element(s) set to .TRUE. If a column of the argument contains only negative numbers, the position(s) of the elements with smallest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose column-wise maximum positions are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each column of the matrix input to the following FORTRAN-PLUS fragment the maximum values(s) in that column are replaced by the value 0.0.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_E_MAX_PR

LM = X05_E_MAX_PR(RM)
RM(LM) = 0.0
RETURN
END
```

15.6 X05_E_MAX_VC

release 1

1 Purpose

X05_E_MAX_VC returns a real vector whose i^{th} component is the maximum value in the i^{th} row of the real matrix argument.

2 Specification

```
REAL VECTOR FUNCTION X05_E_MAX_VC(RM)
REAL RM(,)
```

3 Description

The maximum values are found by locating the position(s) of the maximum value in each row and then taking the value in the first of these positions in each row. These maximum values are then used to construct the required output vector.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose row-wise maximum values are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the routines X05_WEST_BOUNDARY and X05_E_MAX_PC from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each row of the real matrix input to the following FORTRAN-PLUS fragment the maximum value in the row is subtracted from all the values in the row.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)

EXTERNAL REAL VECTOR FUNCTION X05_E_MAX_VC

RM=RM - MATC(X05_E_MAX_VC(RM))
RETURN
END
```

15.7 X05_E_MAX_VR

release 1

1 Purpose

X05_E_MAX_VR returns a real vector whose i^{th} component is the maximum value in the i^{th} column of the real matrix argument.

2 Specification

```
REAL VECTOR FUNCTION X05_E_MAX_VR(RM)
REAL RM(,)
```

3 Description

The maximum values are found by locating the position(s) of the maximum value in each column and then taking the value in the first of these position(s) in each column. These maximum values are then used to construct the required output vector.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose column-wise maximum values are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the routines X05_E_MAX_PR and X05_NORTH_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each column of the real matrix input to the following FORTRAN-PLUS fragment the maximum value in the column is subtracted from all the values in the column.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)

EXTERNAL REAL VECTOR FUNCTION X05_E_MAX_VR

RM=RM - MATR(X05_E_MAX_VR(RM))
RETURN
END
```

15.8 X05_E_MIN_PC

release 1

1 Purpose

X05_E_MIN_PC returns a logical matrix marking the minimum value(s) in each row of the real matrix argument. The i^{th} row of the argument contains one or more elements whose value is the minimum value for the row. The corresponding element(s) of the i^{th} row of the logical matrix are set to .TRUE. to mark that minimum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_E_MIN_PC(RM)
REAL RM(,)
```

3 Description

In each row of the argument which contains only positive numbers the position(s) of the minimum number is found, and the corresponding output logical mask element(s) set to .TRUE. If a row of the argument contains at least one negative number, the position(s) of the negative number with greatest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose row-wise minimum positions are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each row of the matrix input to the following FORTRAN-PLUS fragment the minimum value(s) in that row are replaced by the value 0.0.

```
SUBROUTINE EXAMPLE (RM)
REAL RM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_E_MIN_PC

LM = X05_E_MIN_PC(RM)
RM(LM) = 0.0
RETURN
END
```

15.9 X05_E_MIN_PR

release 1

1 Purpose

X05_E_MIN_PR returns a logical matrix marking the minimum value(s) in each column of the real matrix argument. The i^{th} column of the argument contains one or more elements whose value is the minimum value for the column. The corresponding element(s) of the i^{th} column of the logical matrix are set to .TRUE. to mark that minimum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_E_MIN_PR(RM)
REAL RM(,)
```

3 Description

In each argument column which contains only positive numbers the position(s) of the minimum number is found, and the corresponding output logical mask element(s) set to .TRUE. If a column of the argument contains at least one negative number, the position(s) of the negative number with greatest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose column-wise minimum positions are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each column of the matrix input to the following FORTRAN-PLUS fragment the minimum value(s) in that column are replaced by the value 0.0.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_E_MIN_PR

LM=X05_E_MIN_PR(RM)
RM(LM)=0.0
RETURN
END
```

15.10 X05_E_MIN_VC

release 1

1 Purpose

X05_E_MIN_VC returns a real vector whose i^{th} component is the minimum value in the i^{th} row of the real matrix argument.

2 Specification

```
REAL VECTOR FUNCTION X05_E_MIN_VC(RM)
REAL RM(,)
```

3 Description

The minimum values are found by locating the positions of the minimum values in each row and then taking the value in the first of these positions in each row. The minimum values so found are used to construct the output vector.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose row-wise minimum values are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the routines X05_E_MIN_PC and X05_WEST_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each row of the real matrix input to the following FORTRAN-PLUS fragment the minimum value in the row is subtracted from all the values in the row.

```
SUBROUTINE EXAMPLE (RM)
REAL RM(,)

EXTERNAL REAL VECTOR FUNCTION X05_E_MIN_VC

RM = RM - MATC(X05_E_MIN_VC(RM))
RETURN
END
```

15.11 X05_E_MIN_VR

release 1

1 Purpose

X05_E_MIN_VR returns a real vector whose i^{th} component is the minimum value in the i^{th} column of the real matrix argument.

2 Specification

REAL VECTOR FUNCTION X05_E_MIN_VR (RM)
REAL RM(,)

3 Description

The minimum values are found by locating the positions of the minimum values in each column and then taking the value in the first of these positions in each column. The minimum values so found are used to construct the output vector.

4 References

None

5 Arguments

RM – REAL MATRIX

On entry, RM contains the matrix whose column-wise minimum values are required. RM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the routines X05_E_MIN_PR and X05_NORTH_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each column of the real matrix input to the following FORTRAN-PLUS fragment the minimum value in the column is subtracted from all the values in the column.

```
SUBROUTINE EXAMPLE(RM)
REAL RM(,)

EXTERNAL REAL VECTOR FUNCTION X05_E_MIN_VR

RM = RM - MATR(X05_E_MIN_VR(RM))
RETURN
END
```

15.12 X05_EXCH_P

release 1

1 Purpose

X05_EXCH_P exchanges L planes starting at X with L planes starting at Y, under activity control specified by M. The planes are exchanged in increasing order; you are cautioned about the strange effects which will occur if the two sets of planes overlap.

2 Specification

```
SUBROUTINE X05_EXCH_P(X , Y , M , L)
INTEGER L
LOGICAL M(,)
<any type> X(,),Y(,)
```

3 Description

The areas are exchanged under activity control using a machine code loop.

4 References

None

5 Arguments

X – <*any type*> MATRIX (or MATRIX array)

On entry, X contains the data to be exchanged with Y. On exit, X contains the data originally held in Y.

Y – <*any type*> MATRIX (or MATRIX array)

On entry, Y contains the data to be exchanged with X. On exit, Y contains the data originally held in X.

M – LOGICAL MATRIX

On entry, M defines the mask; .TRUE. indicates elements to be exchanged. M is unchanged on exit.

L – INTEGER

On entry, L specifies the number of planes to be exchanged and must be less than the maximum number of times that a machine code DO-loop may be executed (2^{30} times on the DAP 500). L is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data exchange, planar exchange

11 Example

This FORTRAN-PLUS fragment shows how the routine could be used to exchange two one byte matrices.

```
ENTRY SUBROUTINE SWAP
INTEGER*1 A(,),B(,)
A = 13
B = 25
CALL X05_EXCH_P(A,B,MAT(.TRUE.),8)
TRACE 1 (A, B)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: SWAP at Line 8

Integer Matrix Local Variable A in 8 bits - addressed by Stack + 0.09

(Row 01 Col 01)   25 (* 32)
(Row 02 Col 01)   25 (* 32)
(Row 03 Col 01)   25 (* 32)

    .
    .
    .

(Row 30 Col 01)   25 (* 32)
(Row 31 Col 01)   25 (* 32)
(Row 32 Col 01)   25 (* 32)

Integer Matrix Local Variable B in 8 bits - addressed by Stack + 0.10

(Row 01 Col 01)   13 (* 32)
(Row 02 Col 01)   13 (* 32)
(Row 03 Col 01)   13 (* 32)

    .
    .
    .
```

```
(Row 30 Col 01)   13 (* 32)
(Row 31 Col 01)   13 (* 32)
(Row 32 Col 01)   13 (* 32)

End of Report
```

15.13 X05_GATHER_V_32

release 1

1 Purpose

X05_GATHER_V_32 assigns to the components of a vector the values of those components of a vector array designated by corresponding components of an indexing vector. The index values are interpreted as reduced rank indices to the vector array.

2 Specification

SUBROUTINE X05_GATHER_V_32 (TO , FROM , NFROM , SELECT , IFAIL)

TO and FROM must agree in type and length. They may be INTEGER* < 1 – 4 >, REAL* < 3 – 4 > or CHARACTER. For example:

INTEGER TO ( ) , FROM (, NFROM)
INTEGER NFROM , SELECT ( ) , IFAIL

3 Description

The gathering is performed in a machine code DO loop.

4 References

None

5 Arguments

TO – INTEGER* < 1 – 4 >, REAL* < 3 – 4 > or CHARACTER VECTOR

On exit, TO contains 32 values from array FROM, as selected by SELECT; that is, TO (I) = FROM (SELECT (I)) for I = 1, 32

FROM – INTEGER, REAL or CHARACTER VECTOR array

The dimensions of the array are (, NFROM), agreeing with TO in type and length. FROM is unchanged on exit.

NFROM – INTEGER

The second dimension of array FROM. NFROM is unchanged on exit

SELECT – INTEGER VECTOR

The values are applied as reduced rank indices to array FROM to select values to be assigned to corresponding components of TO. SELECT is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	NFROM was not positive
IFAIL = 2	Values of SELECT were not in range 1 to 32 NFROM

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data manipulation, gather, scatter

11 Example

This FORTRAN-PLUS fragment gathers alternate indexed elements of a 64 element vector into a 32 element vector.

```
      ENTRY SUBROUTINE ENT
      INTEGER FROM(,2),TO(),SELECT()
      DO 10 I=1,64
   10 FROM(I)=10*I
      DO 20 I=1,32
   20 SELECT(I)=2*I
      CALL X05_GATHER_V32(TO,FROM,2,SELECT,IFAIL)
      TRACE 1 (IFAIL)
      TRACE 1 (TO)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 8

Integer Scalar Local Variable IFAIL in 32 bits - on Stack at 0.13

          0

End of Report
```

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 9

Integer Vector Local Variable TO in 32 bits - addressed by Stack + 0.10


 (Component 01)        20,         40,         60,         80,
 (Component 05)       100,        120,        140,        160,
 (Component 09)       180,        200,        220,        240,
 (Component 13)       260,        280,        300,        320,
 (Component 17)       340,        360,        380,        400,
 (Component 21)       420,        440,        460,        480,
 (Component 25)       500,        520,        540,        560,
 (Component 29)       580,        600,        620,        640

End of Report
```

15.14 X05_I_MAX_PC

release 1

1 Purpose

X05_I_MAX_PC returns a logical matrix marking the maximum value(s) in each row of the integer matrix argument. The i^{th} row of the argument contains one or more elements whose value is the maximum value for the row. The corresponding element(s) of the i^{th} row of the logical matrix are set to .TRUE. to mark that maximum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_I_MAX_PC(IM , N)
INTEGER* <N>IM(,)
INTEGER N
```

3 Description

In each row of the argument which contains at least one positive number the position(s) of the maximum positive number is found, and the corresponding output logical mask element(s) set to .TRUE. If a row of the argument contains only negative numbers, the position(s) of the elements with smallest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

IM – INTEGER* <N> MATRIX

On entry, IM contains the matrix whose row-wise maximum positions are required. IM is unchanged on exit.

N – INTEGER

On entry, N specifies the length of the matrix IM in bytes. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each row of the matrix input to the following FORTRAN-PLUS fragment the maximum value(s) in that row are set to zero.

```
SUBROUTINE EXAMPLE(IM)
INTEGER*2 IM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_I_MAX_PC

LM=X05_I_MAX_PC(IM,2)
IM(LM)=0
RETURN
END
```

15.15 X05_I_MAX_PR

release 1

1 Purpose

X05_I_MAX_PR returns a logical matrix marking the maximum value(s) in each column of the integer matrix argument. The i^{th} column of the argument contains one or more elements whose value is the maximum value for the column. The corresponding element(s) of the i^{th} column of the logical matrix are set to .TRUE. to mark that maximum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_I_MAX_PR(IM , N)
INTEGER* <N> IM (,)
INTEGER N
```

3 Description

In each argument column which contains at least one positive number the position(s) of the maximum positive number is found, and the corresponding output logical mask element(s) set to .TRUE. If a column of the argument contains only negative numbers, the position(s) of the elements with smallest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

IM – INTEGER* <N> MATRIX

On entry, IM contains the matrix whose column-wise maximum positions are required. IM is unchanged on exit.

N – INTEGER

On entry, N specifies the length of the matrix IM in bytes. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each column of the matrix input to the following FORTRAN-PLUS fragment the maximum value(s) in that column are set to zero.

```
SUBROUTINE EXAMPLE(IM)
INTEGER*2 IM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_I_MAX_PR

LM=X05_I_MAX_PR(IM,2)
IM(LM)=0
RETURN
END
```

15.16 X05_I_MAX_VC

release 1

1 Purpose

X05_I_MAX_VC returns an integer vector whose i^{th} component is the maximum value in the i^{th} row of the integer matrix argument.

2 Specification

```
INTEGER VECTOR FUNCTION X05_I_MAX_VC(IM)
INTEGER IM(,)
```

3 Description

The maximum values are found by locating the positions of the maximum values in each row and then taking the value in the first of these positions in each row. These maximum values are then used to construct the required output vector.

4 References

None

5 Arguments

IM – INTEGER MATRIX

On entry, IM contains the matrix whose row-wise maximum values are required. IM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the routines X05_I_MAX_PC and X05_WEST_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each row of the integer matrix argument in this FORTRAN-PLUS fragment the maximum value in that row is subtracted from all the values in that row.

```
SUBROUTINE EXAMPLE(IM)
INTEGER IM(,)

EXTERNAL INTEGER VECTOR FUNCTION X05_I_MAX_VC

IM=IM-MATC(X05_I_MAX_VC(IM))
RETURN
END
```

15.17 X05_I_MAX_VR

release 1

1 Purpose

X05_I_MAX_VR returns an integer vector whose i^{th} component is the maximum value in the i^{th} column of the integer matrix argument.

2 Specification

INTEGER VECTOR FUNCTION X05_I_MAX_VR (IM)
INTEGER IM (,)

3 Description

The maximum values are found by locating the positions of the maximum values in each column and then taking the value in the first of these positions in each column. These maximum values are then used to construct the required output vector.

4 References

None

5 Arguments

IM – INTEGER MATRIX

On entry, IM contains the matrix whose column-wise maximum values are required. IM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

This routine calls the routines X05_I_MAX_PR and X05_NORTH_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Maximum

11 Example

In each column of the integer matrix input to the following FORTRAN-PLUS fragment the maximum value in that column is subtracted from all the values in that column.

```
SUBROUTINE EXAMPLE(IM)
INTEGER IM(,)

EXTERNAL INTEGER VECTOR FUNCTION X05_I_MAX_VR

IM=IM-MATR(X05_I_MAX_VR(IM))
RETURN
END
```

15.18 X05_I_MIN_PC

release 1

1 Purpose

X05_I_MIN_PC returns a logical matrix marking the minimum value(s) in each row of the integer matrix argument. The i^{th} row of the argument contains one or more elements whose value is the minimum value for the row. The corresponding element(s) of the i^{th} row of the logical matrix are set to .TRUE. to mark that minimum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_I_MIN_PC(IM , N)
INTEGER* <N> IM(,)
INTEGER N
```

3 Description

In each argument row which contains only positive numbers the position(s) of the minimum number is found, and the corresponding output logical mask element(s) set to .TRUE. If a row of the argument contains at least one negative number, the position(s) of the negative number with greatest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

IM – INTEGER* <N> MATRIX

On entry, IM contains the matrix whose row-wise minimum positions are required. IM is unchanged on exit.

N – INTEGER

On entry, N specifies the length of the matrix IM in bytes. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each row of the matrix input to the following FORTRAN-PLUS fragment the minimum value(s) in that row are set to zero.

```
SUBROUTINE EXAMPLE(IM)
INTEGER*2 IM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_I_MIN_PC

LM=X05_I_MIN_PC(IM,2)
IM(LM)=0
RETURN
END
```

15.19 X05_I_MIN_PR

release 1

1 Purpose

X05_I_MIN_PR returns a logical matrix marking the minimum value(s) in each column of the integer matrix argument. The i^{th} column of the argument contains one or more elements whose value is the minimum value for the column. The corresponding element(s) of the i^{th} column of the logical matrix are set to .TRUE. to mark that minimum value, with all other elements of the logical matrix set to .FALSE.

2 Specification

```
LOGICAL MATRIX FUNCTION X05_I_MIN_PR(IM , N)
INTEGER* <N> IM(,)
INTEGER N
```

3 Description

In each argument column which contains only positive numbers the position(s) of the minimum number is found, and the corresponding output logical mask element(s) set to .TRUE. If a column of the argument contains at least one negative number, the position(s) of the negative number with greatest absolute value are found, and the corresponding logical mask elements set to .TRUE.; all other elements of the output mask are set to .FALSE.

4 References

None

5 Arguments

IM – INTEGER* <N> MATRIX

On entry, IM contains the matrix whose column-wise minimum positions are required. IM is unchanged on exit.

N – INTEGER

On entry, N specifies the length of the matrix IM in bytes. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each column of the input matrix in this FORTRAN-PLUS fragment the minimum value(s) in that column are set to zero.

```
SUBROUTINE EXAMPLE(IM)
INTEGER*2 IM(,)
LOGICAL LM(,)

EXTERNAL LOGICAL MATRIX FUNCTION X05_I_MIN_PR

LM=X05_I_MIN_PR(IM,2)
IM(LM)=0
RETURN
END
```

15.20 X05_I_MIN_VC

release 1

1 Purpose

X05_I_MIN_VC returns an integer vector whose i^{th} component is the minimum value in the i^{th} row of the integer matrix argument.

2 Specification

```
INTEGER VECTOR FUNCTION X05_I_MIN_VC(IM)
INTEGER IM(,)
```

3 Description

The minimum values are found by locating the positions of the minimum values in each row and then taking the value in the first of these positions in each row. The minimum values so found are used to construct the output vector.

4 References

None

5 Arguments

IM – INTEGER MATRIX

On entry, IM contains the matrix whose row-wise minimum values are required. IM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls routines X05_I_MIN_PC and X05_WEST_BOUNDARY from the General Support library.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each row of the integer matrix input to the following FORTRAN-PLUS fragment the minimum value in that row is subtracted from all the values in that row.

```
SUBROUTINE EXAMPLE(IM)
INTEGER IM(,)

EXTERNAL INTEGER VECTOR FUNCTION X05_I_MIN_VC

IM=IM-MATC(X05_I_MIN_VC(IM))
RETURN
END
```

15.21 X05_I_MIN_VR

release 1

1 Purpose

X05_I_MIN_VR returns an integer vector whose i^{th} component is the minimum value in the i^{th} column of the integer matrix argument.

2 Specification

INTEGER VECTOR FUNCTION X05_I_MIN_VR (IM)
INTEGER IM (,)

3 Description

The minimum values are found by locating the positions of the minimum values in each column and then taking the value in the first of these positions in each column. The minimum values so found are used to construct the output vector.

4 References

None

5 Arguments

IM – INTEGER MATRIX

On entry, IM contains the matrix whose column-wise minimum values are required. IM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

The routine calls the General Support library routines X05_I_MIN_PR and X05_NORTH_BOUNDARY.

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Minimum

11 Example

In each column of the integer matrix input to the following FORTRAN-PLUS fragment the minimum value in that column is subtracted from all the values in the column.

```
SUBROUTINE EXAMPLE(IM)
INTEGER IM(,)

EXTERNAL INTEGER VECTOR FUNCTION X05_I_MIN_VR

IM=IM-MATR(X05_I_MIN_VR(IM))
RETURN
END
```

15.22 X05_LOG2

release 1

1 Purpose

X05_LOG2 returns the number of steps required in a recursive doubling algorithm.

2 Specification

```
INTEGER FUNCTION X05_LOG2 (N)
INTEGER N
```

3 Description

The value returned by the routine is:

$$[\log_2(N-1)]+1$$

where square brackets indicate 'integer part of', and N is the input argument.

The routine subtracts 1 from N, then scans the bit pattern of N – 1 serially, starting at the most significant bit, to find the first .TRUE. bit. The required output value equals (11 – the number of serial steps taken).

For N greater than 1024, X05_LOG2 returns an incorrect value, as the routine takes (N modulo 1024) as its argument.

4 References

None

5 Arguments

N – INTEGER

On entry, the value in N should lie in the range 1 – 1024. $N = 0$ will return the result 10; for $N < 0$ the result is undefined. N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Logarithmic algorithm, recursive doubling

11 Example

The example calculates the number of steps required by a recursive doubling algorithm for a problem of size 1001.

Host program

```
      PROGRAM MAIN
      INTEGER N,LOG2N
      COMMON /LOG2N/ N,LOG2N
C
C Initialise data for function
C
      N = 1001
C
C Connect to DAP module
C
      CALL DAPCON('ent.dd')
C
C Send test data to the DAP
C
      CALL DAPSEN('LOG2N',N,1)
C
C Call the DAP ENTRY subroutine
C
      CALL DAPENT('ENT')
C
C Send test data and result from the DAP
C
     CALL DAPREC('LOG2N',N,2)
C
C Release the DAP
C
      CALL DAPREL
C
C Write out the data and result for inspection.
C
      WRITE(6,1) N,LOG2N
1     FORMAT( 'VALUE OF N = ',I6/'STEPS REQUIRED = ',I6)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER N,LOG2N
      COMMON /LOG2N/ N,LOG2N
C
C Note the EXTERNAL statement for this function
C
      EXTERNAL INTEGER SCALAR FUNCTION X05_LOG2
```

```
C
C Convert input data
C
      CALL CONVFSI(N,1)
      LOG2N = X05_LOG2(N)
C
C Convert input data and results back to host format
C
      CALL CONVSFI(N,2)
      RETURN
      END
```

Results

```
VALUE OF N =   1001
STEPS REQUIRED =     10
```

15.23 X05_LONG_INDEX

release 1

1 Purpose

X05_LONG_INDEX generates an integer matrix whose i^{th} element in long vector order is $(i + \mathrm{N} - 1)$, where N is a parameter to the routine.

2 Specification

```
SUBROUTINE X05_LONG_INDEX (IMAT , N)
INTEGER IMAT(,) , N
```

3 Description

The routine calls the FORTRAN-PLUS intrinsic 'Long_Index', and is provided for backwards compatability with existing code.

4 References

None

5 Arguments

IMAT – INTEGER MATRIX

On exit, the i^{th} component in long vector order of IMAT will contain (i + N – 1).

N – INTEGER

On entry, N specifies the value that is required in IMAT(1). N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

Overflow is not detected for large values of N.

10 Keywords

Indexing

11 Example

The example generates a vector indexed from 1 to 1024.

Host program

```
      PROGRAM MAIN
      INTEGER IM(32,32)
      COMMON /IM/IM
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('IM',IM,1024)
      CALL DAPREL
      DO 10 I=1,32
      DO 10 J=1,32
   10 WRITE(6,1000) IM(J,I)
 1000 FORMAT(1X,I6)
      STOP
      END
```

DAP program

```
       ENTRY SUBROUTINE ENT
      INTEGER IM(,)
      COMMON /IM/IM
      CALL X05_LONG_INDEX(IM,1)
      CALL CONVMFI(IM)
      RETURN
      END
```

Results

```
     1
     2
     3
     .
     .
     .
  1024
```

15.24 X05_NORTH_BOUNDARY release 1

1 Purpose

X05_NORTH_BOUNDARY returns a logical matrix containing at most one .TRUE. in each column corresponding to the first .TRUE. (if any) in each column of the logical matrix parameter. That is, the routine is equivalent to the FORTRAN-PLUS code:

```
      KM = .FALSE.
      DO 10 I = 1,32
      IF (.NOT.ANY(LM(,I))) GOTO 10
      KM(,I) = FRST(LM(,I))
   10 CONTINUE
```

2 Specification

```
LOGICAL MATRIX FUNCTION X05_NORTH_BOUNDARY(LM)
LOGICAL LM(,)
```

3 Description

The DAP store plane (logical matrix LM) passed to the routine is treated as a set of 32 logical vectors, arranged so that each vector occupies a complete column. Each of these vectors is dealt with independently, but in parallel.

To each vector is ripple-added a column of all-true bits; the northernmost bit of the vector is treated as least significant. The addition column is thrown away; the column of carry bits from the addition, and a shifted-south version of the column of carries, are XORed to give a vector with only one true element: the northernmost .TRUE. element in each input vector. The 32 resultant vectors, produced in parallel, form the required north boundary matrix.

4 References

None

5 Arguments

LM – LOGICAL MATRIX

On entry, LM is the logical matrix whose north boundary is required. LM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Boundary

11 Example

The following FORTRAN-PLUS fragment takes a 'black and white' logical matrix (a chess board pattern) as input, and returns the north boundary.

```
ENTRY SUBROUTINE ENT
LOGICAL LM(,),KM(,)
EXTERNAL LOGICAL MATRIX FUNCTION X05_NORTH_BOUNDARY
LM=ALTR(1).LEQ.ALTC(1)
KM=X05_NORTH_BOUNDARY(LM)
TRACE 1 (KM)
RETURN
END
```

The result in this case is simply LM .AND. ROWS(1,2)

15.25 X05_PATTERN

release 1

1 Purpose

X05_PATTERN produces four user-selectable patterns, each of which is returned as a logical matrix. The four patterns available are:

0 – The main diagonal
1 – The minor diagonal
2 – A matrix, the rows of which correspond to the rows generated by the built-in function ALTC
3 – The unit lower triangular matrix

2 Specification

LOGICAL MATRIX FUNCTION X05_PATTERN (I)
INTEGER I

3 Description

The routine is provided for backwards compatability with existing code.

4 References

None

5 Arguments

I – INTEGER

On entry I specifies the pattern required. Four values are catered for:

I = 0 : RESULT (J, J) = .TRUE. where $0 < J < 33$; all other elements are .FALSE.

I = 1 : RESULT (J, 33 – J) = .TRUE. where $0 < J < 33$; all other elements are .FALSE.

I = 2 : RESULT (J,) is set equal to the row which generates ALTC (J – 1)

I = 3 : RESULT (J, K) = .TRUE. if J .GE. K where $0 < J, K < 33$

I is unchanged on exit.

6 Error Indicators

If $I < 0$ or $I > 3$ X05_PATTERN returns a logical matrix with all entries .FALSE.

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Pattern generation

11 Example

In the following FORTRAN-PLUS fragment the patterns produced by the routine are used to set up an integer identity matrix and a second matrix having 1 (.TRUE.) below the main diagonal and 0 (.FALSE.) everywhere else.

```
ENTRY SUBROUTINE ENT
INTEGER IDENT(,), LOWER(,)
LOGICAL DIAG(,)
EXTERNAL LOGICAL MATRIX  FUNCTION  X05_PATTERN
DIAG = X05_PATTERN (0)
IDENT = 0
IDENT (DIAG) = 1
LOWER = 0
LOWER(X05_PATTERN(3).AND..NOT.DIAG) = 1
RETURN
END
```

15.26 X05_SCATTER_V_32

release 1

1 Purpose

X05_SCATTER_V_32 takes components of a vector and assigns the values to components of a vector array designated by corresponding components of an indexing vector. The index values are interpreted as reduced rank indices to the vector array.

2 Specification

SUBROUTINE X05_SCATTER_V_32 (FROM , TO , NTO , SELECT , IFAIL)

FROM and TO must agree in type and length. They may be INTEGER* $< 1-4 >$, REAL* $< 3-4 >$ or CHARACTER. For example:

INTEGER FROM ( ) , TO (, NTO)
INTEGER NTO , SELECT ( ) , IFAIL

3 Description

The scattering is performed in a machine code DO loop.

4 References

None

5 Arguments

FROM – INTEGER* $< 1-4 >$, REAL* $< 3-4 >$ or CHARACTER VECTOR

Contains the 32 values to be scattered; it is unchanged on exit.

TO – INTEGER, REAL or CHARACTER VECTOR array

The dimensions of the array are (, NTO), agreeing with FROM in type and length. On exit, TO contains 32 values from FROM, as selected by SELECT;
that is, TO (SELECT (I)) = FROM (I) for I = 1, 32

NTO – INTEGER

The second dimension of array TO; NTO is unchanged on exit

SELECT – INTEGER VECTOR

The values are applied as reduced rank indices to TO, to select components as destinations for corresponding values from array FROM. SELECT is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	NTO was not positive
IFAIL = 2	Values of SELECT were not in range 1 to 32 NTO

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data manipulation, gather, scatter

11 Example

The following FORTRAN-PLUS fragment scatters a 32 element vector to alternate positions in a 64 element vector.

```
      ENTRY SUBROUTINE ENT
      INTEGER FROM(),TO(,2),SELECT()
      DO 10 I=1,64
   10 TO(I)=0
      DO 20 I=1,32
      FROM(I)=I
   20 SELECT(I)=2*I
      CALL X05_SCATTER_V32(FROM,TO,2,SELECT,IFAIL)
      TRACE 1 (IFAIL)
      TRACE 1 (TO)
      RETURN
      END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 9

Integer Scalar Local Variable IFAIL in 32 bits - on Stack at 0.13

          0

End of Report

FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 10

Integer Vector Local Variable TO in 32 bits - addressed by Stack + 0.10
Unconstrained dimensions - 2
```

```
(Element 1)
 (Component 01)        0,        1,        0,        2,
 (Component 05)        0,        3,        0,        4,
 (Component 09)        0,        5,        0,        6,
 (Component 13)        0,        7,        0,        8,
 (Component 17)        0,        9,        0,       10,
 (Component 21)        0,       11,        0,       12,
 (Component 25)        0,       13,        0,       14,
 (Component 29)        0,       15,        0,       16
(Element 2)
 (Component 01)        0,       17,        0,       18,
 (Component 05)        0,       19,        0,       20,
 (Component 09)        0,       21,        0,       22,
 (Component 13)        0,       23,        0,       24,
 (Component 17)        0,       25,        0,       26,
 (Component 21)        0,       27,        0,       28,
 (Component 25)        0,       29,        0,       30,
 (Component 29)        0,       31,        0,       32

End of Report
```

15.27 X05_SHLC_LV

release 1

1 Purpose

X05_SHLC_LV performs a cyclic long vector shift to the left on a number of bit planes, up to a maximum of 256 planes.

2 Specification

```
SUBROUTINE X05_SHLC_LV(V , W , DEPTH , DIST)
INTEGER DEPTH , DIST
LOGICAL V(,,DEPTH) , W(,,DEPTH)
```

3 Description

The shift is carried out in two stages. If the shift distance is D, then North/South shifting is used for that part of the shift given by D modulo 32, and a West shift is used to handle the remaining multiples of 32.

4 References

None

5 Arguments

V – LOGICAL MATRIX array of dimension (,,DEPTH)

On entry, V contains the data to be shifted; V is unchanged on exit.

W – LOGICAL MATRIX array of dimension (,,DEPTH)

On exit, W contains the shifted version of the data in V.

DEPTH – INTEGER

On entry, DEPTH specifies the dimension of V; that is, the number of planes to be shifted (taken modulo 256). DEPTH is unchanged on exit.

DIST – INTEGER

On entry, DIST specifies the magnitude of the shift (taken modulo 1024). DIST is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Shifting

11 Example

The example compares the result from X05_SHLC_LV with that from the built-in function SHLC. The number of positions at which the two results disagree is counted and displayed.

Host program

```
      PROGRAM MAIN
      COMMON /ICOUNT/ICOUNT
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('ICOUNT',ICOUNT,1)
      CALL DAPREL
      WRITE(6,1000) ICOUNT
 1000 FORMAT(' ICOUNT = ',I5)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER IM(,),JM(,),KM(,)
      COMMON /ICOUNT/ICOUNT
      CALL X05_LONG_INDEX(IM,1)
      CALL X05_SHLC_LV(IM,JM,32,99)
      KM=SHLC(IM,99)
      ICOUNT = SUM(KM.NE.JM)
      CALL CONVSFI(ICOUNT,1)
      RETURN
      END
```

Results

```
      ICOUNT = 0
```

15.28 X05_SHLP_LV

release 1

1 Purpose

X05_SHLP_LV performs a planar long vector shift to the left on a number of bit planes, up to a maximum of 256 planes.

2 Specification

```
SUBROUTINE X05_SHLP_LV(V , W , DEPTH , DIST)
INTEGER DEPTH , DIST
LOGICAL V(,,DEPTH) , W(,,DEPTH)
```

3 Description

The shift is carried out in two stages. If the shift distance is D, then North/South shifting is used for that part of the shift given by D modulo 32, and a West shift is used to handle the remaining multiples of 32.

4 References

None

5 Arguments

V – LOGICAL MATRIX array of dimension (,,DEPTH)

On entry, V contains the data to be shifted; V is unchanged on exit.

W – LOGICAL MATRIX array of dimension (,,DEPTH)

On exit, W contains the shifted version of the data in V.

DEPTH – INTEGER

On entry, DEPTH specifies the dimension of V; that is, the number of planes to be shifted (taken modulo 256). DEPTH is unchanged on exit.

DIST – INTEGER

On entry DIST specifies the magnitude of the shift (taken modulo 1024). DIST is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Shifting

11 Example

The example compares the result from X05_SHLP_LV with that from the built-in function SHLP. The number of positions at which the two results disagree is counted and displayed.

Host program

```
      PROGRAM MAIN
      COMMON /ICOUNT/ICOUNT
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('ICOUNT',ICOUNT,1)
      CALL DAPREL
      WRITE(6,1000) ICOUNT
 1000 FORMAT(' ICOUNT =',I5)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER IM(,),JM(,),KM(,)
      COMMON /ICOUNT/ICOUNT
      CALL X05_LONG_INDEX(IM,1)
      CALL X05_SHLP_LV(IM,JM,32,99)
      KM=SHLP(IM,99)
      ICOUNT = SUM(KM.NE.JM)
      CALL CONVSFI(ICOUNT,1)
      RETURN
      END
```

Results

```
      ICOUNT = 0
```

15.29 X05_SHORT_INDEX

release 1

1 Purpose

X05_SHORT_INDEX uses the FORTRAN-PLUS intrinsic routine 'Short_Index', and is provided for backwards compatibility.

2 Specification

```
SUBROUTINE X05_SHORT_INDEX (IVEC , N)
INTEGER IVEC () , N
```

3 Description

The routine is based on the FORTRAN-PLUS intrinsic 'Short_Index'.

4 References

None

5 Arguments

IVEC – INTEGER VECTOR

On exit, the i^{th} component of IVEC will contain $(i + N - 1)$.

N – INTEGER

On entry, N specifies the value that is required in IVEC (1); N is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

Overflow is not detected for extremely large values in N.

10 Keywords

Indexing

11 Example

The example generates a vector indexed from 1 to 32.

Host program

```
      PROGRAM MAIN
      INTEGER IV(32)
      COMMON /IV/IV
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('IV',IV,32)
      CALL DAPREL
      DO 10 I = 1,32
   10 WRITE (6,1000) IV(I)
 1000 FORMAT(1X,I6)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER IV()
      COMMON /IV/IV
      CALL X05_SHORT_INDEX_(IV,1)
      CALL CONVVFI(IV,32,1)
      RETURN
      END
```

Results

```
     1
     2
     3

     .
     .
     .

    32
```

15.30 X05_SHRC_LV

release 1

1 Purpose

X05_SHRC_LV performs a cyclic long vector shift to the right on bit planes, up to a maximum of 256 planes.

2 Specification

```
SUBROUTINE X05_SHRC_LV(V , W , DEPTH , DIST)
INTEGER DEPTH , DIST
LOGICAL V(,,DEPTH) , W(,,DEPTH)
```

3 Description

The shift is carried out in two stages. If the shift distance is D, then North/South shifting is used for that part of the shift given by D modulo 32, and an East shift is used to handle the remaining multiples of 32.

4 References

None

5 Arguments

V – LOGICAL MATRIX array of dimension (,,DEPTH)

On entry, V contains the data to be shifted; V is unchanged on exit.

W – LOGICAL MATRIX array of dimension (,,DEPTH)

On exit, W contains the shifted version of the data in V.

DEPTH – INTEGER

On entry, DEPTH specifies the dimension of V; thgat is, the number of planes to be shifted (taken modulo 256). DEPTH is unchanged on exit.

DIST – INTEGER

On entry, DIST specifies the magnitude of the shift (taken modulo 1024). DIST is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Shifting

11 Example

The example compares the result from X05_SHRC_LV with that from the built-in function SHRC. The number of positions at which the two results disagree is counted and displayed.

Host program

```
      PROGRAM MAIN
      COMMON /ICOUNT/ICOUNT
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('ICOUNT',ICOUNT,1)
      CALL DAPREL
      WRITE(6,1000) ICOUNT
 1000 FORMAT(' ICOUNT =',I5)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER IM(,),JM(,),KM(,)
      COMMON /ICOUNT/ICOUNT
      CALL X05_LONG_INDEX(IM,1)
      CALL X05_SHRC_LV(IM,JM,32,99)
      KM=SHRC(IM,99)
      ICOUNT=SUM(KM.NE.JM)
      CALL CONVSFI(ICOUNT,1)
      RETURN
      END
```

Results

```
      ICOUNT =    0
```

15.31 X05_SHRP_LV

release 1

1 Purpose

X05_SHRP_LV performs a planar long vector shift to the right on a number of bit planes, up to a maximum of 256 planes.

2 Specification

```
SUBROUTINE X05_SHRP_LV(V , W , DEPTH , DIST)
INTEGER DEPTH , DIST
LOGICAL V(,,DEPTH) , W(,,DEPTH)
```

3 Description

The shift is carried out in two stages. If the shift distance is D, then North/South shifting is used for that part of the shift given by D modulo 32, and an East shift is used to handle the remaining multiples of 32.

4 References

None

5 Arguments

V – LOGICAL MATRIX array of dimension (,,DEPTH)

On entry, V contains the data to be shifted; V is unchanged on exit.

W – LOGICAL MATRIX array of dimension (,,DEPTH)

On exit, W contains the shifted version of the data in V.

DEPTH – INTEGER

On entry, DEPTH specifies the dimension of V; that is, the number of planes to be shifted (taken modulo 256). DEPTH is unchanged on exit.

DIST – INTEGER

On entry DIST specifies the magnitude of the shift (taken modulo 1024). DIST is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Shifting

11 Example

The example compares the result from X05_SHRP_LV with that from the built-in function SHRP. The number of positions at which the two results disagree is counted and displayed.

Host program

```
      PROGRAM MAIN
      COMMON /ICOUNT/ICOUNT
      CALL DAPCON('ent.dd')
      CALL DAPENT('ENT')
      CALL DAPREC('ICOUNT',ICOUNT,1)
      CALL DAPREL
      WRITE(6,1000) ICOUNT
 1000 FORMAT(' ICOUNT =',I5)
      STOP
      END
```

DAP program

```
      ENTRY SUBROUTINE ENT
      INTEGER IM(,),JM(,),KM(,)
      COMMON /ICOUNT/ICOUNT
      CALL X05_LONG_INDEX(IM,1)
      CALL X05_SHRP_LV(IM,JM,32,99)
      KM=SHRP(IM,99)
      ICOUNT=SUM(KM.NE.JM)
      CALL CONVSFI(ICOUNT,1)
      RETURN
      END
```

Results

```
      ICOUNT =    0
```

15.32 X05_SOUTH_BOUNDARY release 1

1 Purpose

X05_SOUTH_BOUNDARY returns a logical matrix containing at most one .TRUE. in each column, corresponding to the last .TRUE. (if any) in each column of the logical matrix parameter. That is, the routine is equivalent to the FORTRAN-PLUS code:

```
      KM = .FALSE.
      DO 10 I = 1, 32
      IF (.NOT.ANY (LM (,I))) GOTO 10
      KM(,I) = REV (FRST (REV (LM (,I))))
   10 CONTINUE
```

2 Specification

```
LOGICAL MATRIX FUNCTION X05_SOUTH_BOUNDARY(LM)
LOGICAL LM (,)
```

3 Description

The DAP store plane (logical matrix LM) passed to the routine is treated as a set of 32 logical vectors, arranged so that each vector occupies a complete column. Each of these vectors is dealt with independently, but in parallel.

To each vector is ripple-added a column of all-true bits; the southernmost bit of the vector is treated as least significant. The addition is thrown away; the column of carry bits from the addition, and a shifted-north version of the column of carries, are XORed to give a vector with only one true element: the southernnmost .TRUE. element in each input vector. The 32 resultant vectors, produced in parallel, form the required south boundary matrix.

4 References

None

5 Arguments

LM – LOGICAL MATRIX

On entry, LM is the logical matrix whose south boundary is required. LM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Boundary

11 Example

The following FORTRAN-PLUS fragment takes a 'black and white' logical matrix (a chess board pattern) as input, and returns the south boundary.

```
ENTRY SUBROUTINE ENT
LOGICAL LM(,),KM(,)
EXTERNAL LOGICAL MATRIX FUNCTION X05_SOUTH_BOUNDARY
LM=ALTR(1).LEQ.ALTC(1)
KM=X05_SOUTH_BOUNDARY(LM)
TRACE 1 (KM)
RETURN
END
```

The result in this case is simply LM .AND. ROWS(31,32)

15.33 X05_STRETCH_4

release 1

1 Purpose

X05_STRETCH_4 stretches the first quarter of a real matrix A (considered as a long vector), such that each element is repeated four times consecutively.

2 Specification

REAL MATRIX FUNCTION X05_STRETCH_4(A)
REAL A(,)

3 Description

The routine uses a recursive doubling algorithm to re-arrange the data.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, the first 256 elements of A must be defined. On exit, the 1024 elements of A contain 256 groups of 4 identical elements, the groups being one elements repeated 4 times, from each of the first 256 elements of the input matrix; long vector order is used.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data manipulation

11 Example

The following FORTRAN-PLUS fragment sets up an index matrix such that A (I) = I (I = 1, 2, ... 256), with other elements being undefined. This matrix is then 'stretched' so that:

$$A(I) = \frac{(I-1)}{4}+1 \quad \text{for} \quad I = 1, 2, \ldots 1024$$

```
ENTRY SUBROUTINE ENT
REAL A(,)
INTEGER IM(,)
EXTERNAL REAL MATRIX FUNCTION X05_STRETCH_4
CALL X05_LONG_INDEX(IM,1)
A(ELSL(1,256)) = FLOAT(IM)
A = X05_STRETCH_4(A)
RETURN
END
```

15.34 X05_STRETCH_8

release 1

1 Purpose

X05_STRETCH_8 stretches the first eighth of a real matrix A (considered as a long vector), such that each element is repeated eight times consecutively.

2 Specification

```
REAL MATRIX FUNCTION X05_STRETCH_8(A)
REAL A(,)
```

3 Description

The routine uses a recursive doubling algorithm to re-arrange the data.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, the first 128 elements of A must be defined. On exit, the 1024 elements of A contain 128 groups of 8 identical elements, the groups being one elements repeated 8 times, from each of the first 128 elements of the input matrix; long vector order is used.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data manipulation

11 Example

The following FORTRAN-PLUS fragment sets up an index matrix such that $A(I) = I$ $(I = 1, 2, \ldots 128)$, with other elements being undefined. This matrix is then 'stretched' so that:

$$A(I) = \frac{(I-1)}{8} + 1 \quad \text{for} \quad I = 1, 2, \ldots 1024$$

```
ENTRY SUBROUTINE ENT
REAL A(,)
INTEGER IM(,)
EXTERNAL REAL MATRIX FUNCTION X05_STRETCH_8
CALL X05_LONG_INDEX(IM,1)
A(ELSL(1,128)) = FLOAT(IM)
A = X05_STRETCH_8(A)
RETURN
END
```

15.35 X05_STRETCH _N

release 1

1 Purpose

X05_STRETCH _N stretches the first n^{th} of a real matrix A (considered as a long vector), such that each element is repeated n times consecutively ($n = 2^{\mathrm{I}}$), I being a positive integer.

2 Specification

REAL MATRIX FUNCTION X05_STRETCH _N (A, I, IFAIL)
REAL A (,)

3 Description

The routine uses a recursive doubling algorithm to re-arrange the data.

4 References

None

5 Arguments

A – REAL MATRIX

On entry, the first $1024/n$ elements of A must be defined. On exit, the 1024 elements of A contain $1024/n$ groups of n identical elements, the groups being one element repeated n times, from each of the first $1024/n$ elements of the input matrix; long vector order is used.

I – INTEGER

I is the power of 2, such that $n = 2^{\mathrm{I}}$. I is unchanged on exit

IFAIL – INTEGER

On exit, IFAIL = 1 if the implied value of n is greater than 32

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Data manipulation

11 Example

The following FORTRAN-PLUS fragment sets up an index matrix such that A (I) = I (I = 1, 2, ... 128), with other elements being undefined. This matrix is then 'stretched' so that:

$$A\,(I) = \frac{(I-1)}{8} + 1 \qquad \text{for} \quad I = 1, 2, \ldots 1024$$

```
ENTRY SUBROUTINE ENT
REAL A(,)
INTEGER IM(,)
EXTERNAL REAL MATRIX FUNCTION X05_STRETCH_N
CALL X05_LONG_INDEX(IM,1)
A(ELSL(1,128)) = FLOAT(IM)
A =  X05_STRETCH_N( A,3,IFAIL )
trace 1(a)
RETURN
END
```

Results

```
FORTRAN-PLUS Trace
FORTRAN-PLUS Subroutine: ENT at Line 8

Real Matrix Local Variable A in 32 bits - addressed by Stack + 0.09

(Row 01 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 02 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 03 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 04 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 05 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 06 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 07 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 08 Col 01)  1.0000000E+00,  5.0000000E+00,  9.0000000E+00,
(Row 09 Col 01)  2.0000000E+00,  6.0000000E+00,  1.0000000E+01,
(Row 10 Col 01)  2.0000000E+00,  6.0000000E+00,  1.0000000E+01,
(Row 11 Col 01)  2.0000000E+00,  6.0000000E+00,  1.0000000E+01,

plus rest of TRACE output...
```

15.36 X05_SUM_LEFT_I2

release 1

1 Purpose

X05_SUM_LEFT_I2 takes as input the long vector A (INTEGER* 2) and returns an (INTEGER* 2) long vector, each of whose elements is the sum of all the elements to the left of the corresponding element of A, excluding the element itself.

2 Specification

```
INTEGER* 2 MATRIX FUNCTION X05_SUM_LEFT_I2 (A)
INTEGER* 2 A (,)
```

3 Description

Let A $(= A_{ij})$ be the given long vector. The required long vector result S $(= S_{ij})$ is given by:

$$S_{ij} = \sum_{k=1}^{j-1}\sum_{l=1}^{32} A_{lk} + \sum_{k=1}^{i-1} A_{kj}$$

The sum is broken down into the following steps:

1 $B_{ij} = \sum_{k=1}^{i} A_{kj}$ the cumulative sums down each column

2 $C_{ij} = B_{32,j-1}$ for each i, where $B_{32,0} = 0$

3 $D_{ij} = \sum_{k=1}^{j} C_{ik}$ the cumulative sums of C along each row

4 $S_{ij} = D_{ij} + B_{i-1,j}$ where $B_{0,j} = 0$

The summations (1) and (3) are performed using standard parallel algorithms (6 steps). The remaining operations consist of shifts and a matrix add.

4 References

None

5 Arguments

A – INTEGER* 2

On entry, A contains the long vector on which the sum left is to be performed. A is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

The results are accurate provided there is no overflow.

9 Further Comments

None

10 Keywords

None

11 Example

In the example, a sum-left is performed on an integer long vector with all components equal to 1. The first five and last five values of the input and resulting long vectors are printed, in long vector order.

Host program

```
      PROGRAM HTSL2

      INTEGER*2 ILV1(32,32),ILV2(32,32)
      COMMON /BDATA/ILV1,ILV2

      CALL DAPCON('tsl2.dd')
      CALL DAPENT('TSL2')

      CALL DAPREC('BDATA',ILV1,1024)
      CALL DAPREL

      WRITE(6,6001)
      WRITE(6,6002) (ILV1(I,1),I=1,5),(ILV1(I,32),I=28,32)
      WRITE(6,6003)
      WRITE(6,6004) (ILV2(I,1),I=1,5),(ILV2(I,32),I=28,32)
 6001 FORMAT('INPUT VECTOR'/)
 6002 FORMAT(5(1X,I1),' . . .',5(1X,I1))
 6003 FORMAT(//,'RESULT'/)
 6004 FORMAT(5(1X,I1),' . . .',5(1X,I4))

      STOP
      END
```

DAP program

```
ENTRY SUBROUTINE TSL2

INTEGER*2 ILV1(,),ILV2(,)
COMMON /BDATA/ILV1,ILV2

EXTERNAL INTEGER*2 MATRIX FUNCTION X05_SUM_LEFT_I2

ILV1=1
ILV2=X05_SUM_LEFT_I2(ILV1)

CALL CONVMF2(ILV1)
CALL CONVMF2(ILV2)

RETURN
END
```

Results

```
INPUT VECTOR

 1 1 1 1 1 . . . 1 1 1 1 1

RESULT

 0 1 2 3 4 . . . 1019 1020 1021 1022 1023
```

15.37 X05_SUM_RIGHT_I2

release 1

1 Purpose

X05_SUM_RIGHT_I2 takes as input the long vector A (INTEGER* 2) and returns an (INTEGER* 2) long vector each of whose elements is the sum of all the elements on the right of the corresponding element of A. The sum is strict in the sense that the element itself is not included.

2 Specification

INTEGER* 2 MATRIX FUNCTION X05_SUM_RIGHT_I2 (A)
INTEGER* 2 A (,)

3 Description

Let A $(= A_{ij})$ be the given long vector. The required long vector result S $(= S_{ij})$ is given by:

$$S_{ij} = \sum_{k=j+1}^{32} \sum_{l=i+1}^{32} A_{lk} + \sum_{k=j+1}^{32} A_{kj}$$

The sum is broken down into the following steps:

1 $B_{ij} = \sum_{l=i+1}^{32} A_{lj}$ the cumulative sums up each column

2 $C_{ij} = B_{32,j+1}$ for each i, where $B_{i,33} = 0$

3 $D_{ij} = \sum_{k=j+1}^{32} C_{ik}$ the cumulative sums of C along each row (right to left)

4 $S_{ij} = D_{ij} + B_{i,j+1}$ where $B_{i,33} = 0$

The summations (1) and (3) are performed using the standard parallel algorithms (6 steps). The remaining operations consist of shifts and a matrix add.

4 References

None

5 Arguments

A – INTEGER* 2

On entry, A contains the long vector on which the sum-right is to be performed. A is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Further Comments

None

9 Keywords

None

10 Example

In the example, a sum-right is performed on an integer vector with all components equal to 1. The first five and last five values of the input and resulting long vectors are printed in long vector order.

Host program

```
      PROGRAM HTSR2

      INTEGER*2 ILV1(32,32),ILV2(32,32)
      COMMON /BDATA/ILV1,ILV2

      CALL DAPCON('tsr2.dd')
      CALL DAPENT('TSR2')

      CALL DAPREC('BDATA',ILV1,1024)
      CALL DAPREL

      WRITE(6,6001)
      WRITE(6,6002) (ILV1(I,1),I=1,5),(ILV1(I,32),I=28,32)
      WRITE(6,6003)
      WRITE(6,6004) (ILV2(I,1),I=1,5),(ILV2(I,32),I=28,32)
 6001 FORMAT('INPUT VECTOR'/)
 6002 FORMAT(5(1X,I1),' . . .',5(1X,I1))
 6003 FORMAT(//,'RESULT'/)
 6004 FORMAT(5(1X,I4),' . . .',5(1X,I1))

      STOP
      END
```

DAP program

```
ENTRY SUBROUTINE TSR2

INTEGER*2 ILV1(,),ILV2(,)
COMMON /BDATA/ILV1,ILV2

EXTERNAL INTEGER*2 MATRIX FUNCTION X05_SUM_RIGHT_I2

ILV1=1
ILV2=X05_SUM_RIGHT_I2(ILV1)

CALL CONVMF2(ILV1)
CALL CONVMF2(ILV2)

RETURN
END
```

Results

```
INPUT VECTOR

 1 1 1 1 1 . . . 1 1 1 1 1


RESULT

 1023 1022 1021 1020 1019 . . . 4 3 2 1 0
```

15.38 X05_UNCRINKLE

release 1

1 Purpose

X05_UNCRINKLE effects a transformation in data storage format for vertical mode data occupying an array of matrices from 'crinkled' to 'sliced' storage.

2 Specification

SUBROUTINE X05_UNCRINKLE (S, L, NR, NC, IFAIL)
<*any type, any length*> S (, , NR, NC)
INTEGER BL, NR, NC, IFAIL

3 Description

The data is conceptually considered to occupy an array C of components of size 32 NR by 32 NC. (NR or NC are positive integers, not excluding 1). The storage area, S, is an NR by NC array of matrices. In the 'sliced' format:

$$S(i_r, i_c, j_r, j_c) = C(i_r+32(j_r-1), i_c+32(j_c-1))$$

that is, each value of j_r selects a contiguous group of 32 rows of C, and so on.

In the 'crinkled' format:

$$S(i_r, i_c, j_r, j_c) = C(j_r+\mathrm{NR}\, i_r-1), j_c+\mathrm{NC}(i_c-1)))$$

that is, each value of i_r selects a contiguous group of NR rows of C, and so on.

In the 'sliced' format the conceptual array is divided into subarrays of size 32 by 32. In the 'crinkled' format the conceptual array is divided into subarrays of size NR by NC.

To carry out the transformation, first a mapping transformation is done on East – West vertical sections of the data area. Each section is regarded as an array of 32 NC data items; each item is of length L by NR (vertical) bits. The transformation reverses the mapping order so that succesive horizontal sets of NC data items are rethreaded vertically.

Then a similar transformation is done on NC separate groups of North – South vertical sections of the data area. Each section of each group is regarded as an array of 32 NR data items; each item is of length L (vertical) bits. The transformation reverses the mapping order so that successive horizontal sets of NR data items are rethreaded vertically.

4 References

None

5 Arguments

S – <*any type, any length*> MATRIX array of dimension (, , NR, NC)

On entry, S contains the sliced data to be reformatted. On exit, S contains the data in crinkled form.

L – INTEGER

On entry, L specifies the length in bits of the components of S; L is unchanged on exit.

NR – INTEGER

On entry, NR specifies the first unconstrained dimension of S; NR is unchanged on exit.

5 Arguments – continued

NC – INTEGER

On entry, NC specifies the second unconstrained dimension of S; NC is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see **Error Indicators** below) IFAIL contains zero on exit.

6 Error Indicators

Errors detected by the routine:

IFAIL = 1	either NR or NC was less than 1
IFAIL = 2	L was less than 1

7 Auxiliary Routines

to be supplied ...

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Crinkled data storage, data formatting, data movement, sliced data storage

11 Example

The following FORTRAN-PLUS fragment shows how the routine can be used in an entry subroutine to convert a matrix set from crinkled to sliced form.

```
      ENTRY SUBROUTINE ENT
      REAL A(,,2,2)
      COMMON /A/A
      DO 10 I=1,2
      DO 10 J=1,2
      CALL CONVFM4(A(,,I,J))
   10 CONTINUE
      CALL X05_UNCRINKLE(A,4,2,2,IFAIL)
      IF (IFAIL.NE.0) RETURN
C     DAP processing
      RETURN
      END
```

15.39 X05_WEST_BOUNDARY

release 1

1 Purpose

X05_WEST_BOUNDARY returns a logical matrix containing at most one .TRUE. element in each row corresponding to the first .TRUE. (if any) in each row of the logical matrix parameter. That is, the subroutine is equivalent to the FORTRAN-PLUS code:

```
      KM = .FALSE.
      DO 10 I = 1, 32
      IF (.NOT. ANY (LM (I,))) GOTO 10
      KM (I,) = FRST (LM (I,))
10    CONTINUE
```

2 Specification

```
LOGICAL MATRIX FUNCTION X05_WEST_BOUNDARY(LM)
LOGICAL LM (,)
```

3 Description

The DAP store plane (logical matrix LM) passed to the routine is treated as a set of 32 logical vectors, arranged so that each vector occupies a complete row. Each of these vectors is dealt with independently, but in parallel.

To each vector is ripple-added a row of all-true bits; the westernmost bit of the vector is treated as least significant. The addition is thrown away; the row of carry bits from the addition, and a shifted-east version of the row of carries, are XORed to give a vector with only one true element: the westernnmost .TRUE. element in each input vector. The 32 resultant vectors, produced in parallel, form the required west boundary matrix.

4 References

None

5 Arguments

LM – LOGICAL MATRIX

On entry, LM is the logical matrix whose west boundary is required. LM is unchanged on exit.

6 Error Indicators

None

7 Auxiliary Routines

None

8 Accuracy

Not applicable

9 Further Comments

None

10 Keywords

Boundary

11 Example

The following FORTRAN-PLUS fragment takes a 'black and white' logical matrix (a chess board pattern) as input and returns the west boundary.

```
ENTRY SUBROUTINE ENT
LOGICAL LM(,),KM(,)
EXTERNAL LOGICAL MATRIX FUNCTION X05_WEST_BOUNDARY
LM=ALTR(1).LEQ.ALTC(1)
KM=X05_WEST_BOUNDARY(LM)
TRACE 1 (KM)
RETURN
END
```

The result in this case is simply LM .AND. COLS(1,2)