NAG Library Routine Document

M01EDF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

M01EDF rearranges a vector of complex numbers into the order specified by a vector of ranks.

2 Specification

```
SUBROUTINE M01EDF (CV, M1, M2, IRANK, IFAIL)

INTEGER                 M1, M2, IRANK(M2), IFAIL
COMPLEX (KIND=nag_wp) CV(M2)
```

3 Description

M01EDF is designed to be used typically in conjunction with the M01D ranking routines. After one of the M01D routines has been called to determine a vector of ranks, M01EDF can be called to rearrange a vector of complex numbers into the rank order. If the vector of ranks has been generated in some other way, then M01ZBF should be called to check its validity before M01EDF is called.

4 References

None.

5 Parameters

1: CV(M2) – COMPLEX (KIND=nag_wp) array *Input/Output*

On entry: elements M1 to M2 of CV must contain complex values to be rearranged.

On exit: these values are rearranged into rank order. For example, if IRANK(i) = M1, then the initial value of CV(i) is moved to CV(M1).

2: M1 – INTEGER *Input*

3: M2 – INTEGER *Input*

On entry: M1 and M2 must specify the range of the ranks supplied in IRANK and the elements of CV to be rearranged.

Constraint: $0 < \text{M1} \leq \text{M2}$.

4: IRANK(M2) – INTEGER array *Input/Output*

On entry: elements M1 to M2 of IRANK must contain a permutation of the integers M1 to M2, which are interpreted as a vector of ranks.

On exit: used as internal workspace prior to being restored and hence is unchanged.

5: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then

the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, M2 < 1,
or M1 < 1,
or M1 $>$ M2.

IFAIL $= 2$

Elements M1 to M2 of IRANK contain a value outside the range M1 to M2.

IFAIL $= 3$

Elements M1 to M2 of IRANK contain a repeated value.

If IFAIL $= 2$ or 3, elements M1 to M2 of IRANK do not contain a permutation of the integers M1 to M2. On exit, the contents of CV may be corrupted. To check the validity of IRANK without the risk of corrupting CV, use M01ZBF.

7 Accuracy

Not applicable.

8 Further Comments

The average time taken by the routine is approximately proportional to n, where $n = \text{M2} - \text{M1} + 1$.

9 Example

This example reads a matrix of complex numbers and rearranges its rows so that the elements in the kth column are in ascending order of modulus. To do this, the program first calls M01DAF to rank the moduli of the elements in the kth column, and then calls M01EDF to rearrange each column into the order specified by the ranks. The value of k is read from the datafile.

9.1 Program Text

```
    Program m01edfe

!     M01EDF Example Program Text

!     Mark 24 Release. NAG Copyright 2012.

!     .. Use Statements ..
      Use nag_library, Only: m01daf, m01edf, nag_wp, x04daf
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                           :: i, ifail, j, k, m1, m2, n
```

```
      Character (30)                   :: string
!     .. Local Arrays ..
      Complex (Kind=nag_wp), Allocatable :: cm(:,:)
      Real (Kind=nag_wp), Allocatable  :: cmod(:)
      Integer, Allocatable             :: irank(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: abs
!     .. Executable Statements ..
      Write (nout,*) 'M01EDF Example Program Results'
      Flush (nout)

!     Skip heading in data file
      Read (nin,*)

      Read (nin,*) m2, n, k

      If (k<1 .Or. k>n) Then
        Go To 100
      End If

      Allocate (cm(m2,n),cmod(m2),irank(m2))

      m1 = 1

      Do i = m1, m2
        Read (nin,*)(cm(i,j),j=1,n)
      End Do

!     Calculate the moduli of the elements in the K-th column.

      Do i = m1, m2
        cmod(i) = abs(cm(i,k))
      End Do

!     Rearrange the rows so that the elements in the K-th column
!     are in ascending order of modulus.

      ifail = 0
      Call m01daf(cmod,m1,m2,'Ascending',irank,ifail)

!     Rearrange each column into the order specified by IRANK.

      Do j = 1, n

        ifail = 0
        Call m01edf(cm(m1,j),m1,m2,irank,ifail)

      End Do

!     Print the results.

      Write (nout,*)
      Write (string,99999) 'Matrix sorted on column', k
      Flush (nout)

      ifail = 0
      Call x04daf('General',' ',m2-m1+1,n,cm(m1,1),m2,string,ifail)

100   Continue

99999 Format (1X,A,I3)
    End Program m01edfe
```

9.2 Program Data

```
M01EDF Example Program Data
12 3 2
(6.0, 1.0) (5.0,-2.0) (4.0, 4.0)
(5.0,-3.0) (2.0,-2.0) (1.0, 1.0)
(2.0, 2.0) (4.0, 1.0) (9.0,-3.0)
```

```
(4.0, 2.0) (9.0, 6.0) (6.0, 4.0)
(4.0, 0.0) (9.0, 3.0) (5.0, 1.0)
(4.0,-8.0) (1.0, 5.0) (2.0, 1.0)
(3.0,-3.0) (4.0,-5.0) (1.0, 0.0)
(2.0, 4.0) (4.0,-2.0) (6.0,-1.0)
(1.0, 1.0) (6.0, 1.0) (4.0, 0.0)
(9.0, 1.0) (3.0, 3.0) (2.0,-4.0)
(6.0,-1.0) (2.0, 3.0) (5.0,-3.0)
(4.0,-5.0) (9.0, 9.0) (6.0, 7.0)
```

9.3 Program Results

```
 M01EDF Example Program Results

  Matrix sorted on column  2
              1           2           3
 1      5.0000      2.0000      1.0000
       -3.0000     -2.0000      1.0000

 2      6.0000      2.0000      5.0000
       -1.0000      3.0000     -3.0000

 3      2.0000      4.0000      9.0000
        2.0000      1.0000     -3.0000

 4      9.0000      3.0000      2.0000
        1.0000      3.0000     -4.0000

 5      2.0000      4.0000      6.0000
        4.0000     -2.0000     -1.0000

 6      4.0000      1.0000      2.0000
       -8.0000      5.0000      1.0000

 7      6.0000      5.0000      4.0000
        1.0000     -2.0000      4.0000

 8      1.0000      6.0000      4.0000
        1.0000      1.0000      0.0000

 9      3.0000      4.0000      1.0000
       -3.0000     -5.0000      0.0000

10      4.0000      9.0000      5.0000
        0.0000      3.0000      1.0000

11      4.0000      9.0000      6.0000
        2.0000      6.0000      4.0000

12      4.0000      9.0000      6.0000
       -5.0000      9.0000      7.0000
```