NAG Library Routine Document

M01DJF

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

M01DJF ranks the columns of a matrix of real numbers in ascending or descending order.

2 Specification

```
SUBROUTINE M01DJF (RM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
INTEGER            LDM, M1, M2, N1, N2, IRANK(N2), IFAIL
REAL (KIND=nag_wp) RM(LDM,N2)
CHARACTER(1)       ORDER
```

3 Description

M01DJF ranks columns N1 to N2 of a matrix, using the data in rows M1 to M2 of those columns. The ordering is determined by first ranking the data in row M1, then ranking any tied columns according to the data in row $M1 + 1$, and so on up to row M2.

M01DJF uses a variant of list-merging, as described on pages 165–166 in Knuth (1973). The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal columns preserve their ordering in the input data.

4 References

Knuth D E (1973) *The Art of Computer Programming (Volume 3)* (2nd Edition) Addison–Wesley

5 Parameters

1: RM(LDM, N2) – REAL (KIND=nag_wp) array *Input*

On entry: rows M1 to M2 of columns N1 to N2 of RM must contain real data to be ranked.

2: LDM – INTEGER *Input*

On entry: the first dimension of the array RM as declared in the (sub)program from which M01DJF is called.

Constraint: $\text{LDM} \geq \text{M2}$.

3: M1 – INTEGER *Input*

On entry: the index of the first row of RM to be used.

Constraint: $\text{M1} > 0$.

4: M2 – INTEGER *Input*

On entry: the index of the last row of RM to be used.

Constraint: $\text{M2} \geq \text{M1}$.

5: N1 – INTEGER *Input*

On entry: the index of the first column of RM to be ranked.

Constraint: $\mathrm{N1} > 0$.

6: N2 – INTEGER *Input*

On entry: the index of the last column of RM to be ranked.

Constraint: $\mathrm{N2} \geq \mathrm{N1}$.

7: ORDER – CHARACTER(1) *Input*

On entry: if ORDER = 'A', the columns will be ranked in ascending (i.e., nondecreasing) order.

If ORDER = 'D', into descending order.

Constraint: ORDER = 'A' or 'D'.

8: IRANK(N2) – INTEGER array *Output*

On exit: elements N1 to N2 of IRANK contain the ranks of the corresponding columns of RM. Note that the ranks are in the range N1 to N2: thus, if the ith column of RM is the first in the rank order, IRANK(i) is set to N1.

9: 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, $\mathrm{M2} < 1$,
or $\mathrm{N2} < 1$,
or $\mathrm{M1} < 1$,
or $\mathrm{M1} > \mathrm{M2}$,
or $\mathrm{N1} < 1$,
or $\mathrm{N1} > \mathrm{N2}$,
or $\mathrm{LDM} < \mathrm{M2}$.

IFAIL = 2

On entry, ORDER is not ‘A’ or ‘D’.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The average time taken by the routine is approximately proportional to $n \times \log(n)$, where $n = \mathrm{N2} - \mathrm{N1} + 1$.

10 Example

This example reads a matrix of real numbers and ranks the columns in ascending order.

10.1 Program Text

```
    Program m01djfe

!     M01DJF Example Program Text

!     Mark 25 Release. NAG Copyright 2014.

!     .. Use Statements ..
      Use nag_library, Only: m01djf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                          :: i, ifail, j, ldm, m1, m2, n1, n2
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: rm(:,:)
      Integer, Allocatable             :: irank(:)
!     .. Executable Statements ..
      Write (nout,*) 'M01DJF Example Program Results'

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

      Read (nin,*) m2, n2
      ldm = m2
      Allocate (rm(ldm,n2),irank(n2))

      m1 = 1
      n1 = 1

      Do i = m1, m2
        Read (nin,*)(rm(i,j),j=n1,n2)
      End Do

      ifail = 0
      Call m01djf(rm,ldm,m1,m2,n1,n2,'Ascending',irank,ifail)

      Write (nout,*)
      Write (nout,*) 'Data'
      Write (nout,*)

      Do i = m1, m2
        Write (nout,99999)(rm(i,j),j=n1,n2)
      End Do

      Write (nout,*)
      Write (nout,*) 'Ranks'
      Write (nout,*)
      Write (nout,99998)(irank(i),i=n1,n2)

99999 Format (1X,12F6.1)
99998 Format (1X,12I6)
    End Program m01djfe
```

10.2 Program Data

```
M01DJF Example Program Data
3 12
5.0 4.0 3.0 2.0 2.0 1.0 9.0 4.0 4.0 2.0 2.0 1.0
3.0 8.0 2.0 5.0 5.0 6.0 9.0 8.0 9.0 5.0 4.0 1.0
9.0 1.0 6.0 1.0 2.0 4.0 8.0 1.0 2.0 2.0 6.0 2.0
```

10.3 Program Results

```
 M01DJF Example Program Results

 Data

    5.0   4.0   3.0   2.0   2.0   1.0   9.0   4.0   4.0   2.0   2.0   1.0
    3.0   8.0   2.0   5.0   5.0   6.0   9.0   8.0   9.0   5.0   4.0   1.0
    9.0   1.0   6.0   1.0   2.0   4.0   8.0   1.0   2.0   2.0   6.0   2.0

 Ranks

     11     8     7     4     5     2    12     9    10     6     3     1
```