NAG Library Routine Document

M01ZAF

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

M01ZAF inverts a permutation, and hence converts a rank vector to an index vector, or vice versa.

2 Specification

```
SUBROUTINE M01ZAF (IPERM, M1, M2, IFAIL)
INTEGER IPERM(M2), M1, M2, IFAIL
```

3 Description

There are two common ways of describing a permutation using an integer vector IPERM. The first uses ranks: IPERM(i) holds the position to which the ith data element should be moved in order to sort the data; in other words its rank in the sorted order. The second uses indices: IPERM(i) holds the current position of the data element which would occur in ith position in sorted order. For example, given the values

3.5 5.9 2.9 0.5

to be sorted in ascending order, the ranks would be

3 4 2 1

and the indices would be

4 3 1 2

The M01D routines generate ranks, and the M01E routines require ranks to be supplied to specify the reordering. However if it is desired simply to refer to the data in sorted order without actually reordering them, indices are more convenient than ranks (see the example in Section 10).

M01ZAF can be used to convert ranks to indices, or indices to ranks, as the two permutations are inverses of each another.

4 References

None.

5 Parameters

1: IPERM(M2) – INTEGER array *Input/Output*

On entry: elements M1 to M2 of IPERM must contain a permutation of the integers M1 to M2.

On exit: these elements contain the inverse permutation of the integers M1 to M2.

2: M1 – INTEGER *Input*

3: M2 – INTEGER *Input*

On entry: M1 and M2 must specify the range of elements used in the array IPERM and the range of values in the permutation, as specified under IPERM.

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

4: 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 IPERM contain a value outside the range M1 to M2.

IFAIL $= 3$

Elements M1 to M2 of IPERM contain a repeated value.

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.

If IFAIL $= 2$ or 3, elements M1 to M2 of IPERM do not contain a permutation of the integers M1 to M2; on exit these elements are usually corrupted. To check the validity of a permutation without the risk of corrupting it, use M01ZBF.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example reads a matrix of real numbers and prints its rows in ascending order as ranked by M01DEF. The program first calls M01DEF to rank the rows, and then calls M01ZAF to convert the rank vector to an index vector, which is used to refer to the rows in sorted order.

10.1 Program Text

```
    Program m01zafe

!     M01ZAF Example Program Text

!     Mark 25 Release. NAG Copyright 2014.

!     .. Use Statements ..
      Use nag_library, Only: m01def, m01zaf, 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             :: iperm(:)
!     .. Executable Statements ..
      Write (nout,*) 'M01ZAF Example Program Results'

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

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

      m1 = 1
      n1 = 1

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

      ifail = 0
      Call m01def(rm,ldm,m1,m2,n1,n2,'Ascending',iperm,ifail)

      ifail = 0
      Call m01zaf(iperm,m1,m2,ifail)

      Write (nout,*)
      Write (nout,*) 'Matrix sorted by rows'
      Write (nout,*)

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

99999 Format (1X,3F7.1)
    End Program m01zafe
```

10.2 Program Data

```
M01ZAF Example Program Data
12 3
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0
```

10.3 Program Results

```
 M01ZAF Example Program Results

 Matrix sorted by rows

     1.0    6.0    4.0
     2.0    4.0    6.0
     2.0    4.0    9.0
     3.0    4.0    1.0
     4.0    1.0    2.0
     4.0    9.0    5.0
     4.0    9.0    6.0
     4.0    9.0    6.0
     5.0    2.0    1.0
     6.0    2.0    5.0
     6.0    5.0    4.0
     9.0    3.0    2.0
```