1 SUMMARY

This subroutine uses a variant of Sloan's algorithm to **generate an element assembly ordering** that is efficient when subsequently used with a frontal solver (for example, the packages MA42 and MA62). The number of floating-point operations and the storage required by a frontal solver for an unassembled finite-element matrix are dependent upon the order in which the elements are assembled; the variation in the performance of different element orderings can be significant. The assembly ordering obtained by MC63 is designed to reduce the maximum and root-mean-square (r.m.s.) wavefronts and the profile, which in turn reduce storage requirements and computation times for the frontal solver. Only the pattern of the finite elements is used.

Let n denote the number of variables (degrees of freedom) in the finite-element mesh. If f_i denotes the number of variables in the front before the ith elimination, the maximum wavefront is defined as

$$\max_{1\le i\le n}\{f_i\},$$

the profile is defined as

$$\sum_{i=1}^{n} f_i,$$

and the root-mean-square wavefront is defined as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n} f_i^2}.$$

The user can choose between using a direct and an indirect element ordering algorithm. The indirect element ordering algorithm first orders the variables and then orders the elements. An option exists for working with supervariables in place of variables (a supervariable is a collection of one or more variables that belong to the same set of finite elements). If the problem has significantly fewer supervariables than variables, using supervariables will reduce the execution time of the ordering algorithm and will, in general, produce an ordering of comparable quality. Moreover, if the number of elements is less than the number of supervariables, it is generally quicker to order the elements using the direct ordering algorithm. However, for a given problem it is difficult to predict whether an indirect or a direct element ordering will produce the smaller wavefront.

ATTRIBUTES — Version: 1.0.0. (12 July 2004) **Types:** Real (single, double). **Remark:** Supersedes MC43. **Calls:** MC60. **Original date:** March 1998. **Origin:** J. A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Argument lists and calling sequence

There are three entries:

(a) MC63I/ID sets default values for control parameters. It should normally be called once prior to calling MC63A/AD.

(b) MC63A/AD reorders the elements.

(c) MC63B/BD computes, for a given element assembly order, the maximum front size, the profile, and the root-mean-square wavefront.

Only MC63A/AD provides extensive checks on the data. MC63B/BD optionally checks for duplicate and out-of-range entries. MC63B/BD is called by MC63A/AD but may also be used in combination with another algorithm for choosing an element assembly order.

2.1.1 To set default values for the control parameters

The single precision version

```
CALL MC63I(ICNTL)
```

The double precision version

```
CALL MC63ID(ICNTL)
```

ICNTL is an INTEGER array of length 10 that need not be set on entry. This array is used to hold control parameters. On return, ICNTL contains default values. If the user wishes to use values other than the defaults, the corresponding entries in ICNTL should be reset after the call to MC63I/ID. The control parameters are:

ICNTL(1) is the stream number for error messages and has the default value 6. Printing of error messages is suppressed if ICNTL(1) $\leq$ 0.

ICNTL(2) is the stream number for warning messages and has the default value 6. Printing of warning messages is suppressed if ICNTL(2) $\leq$ 0.

ICNTL(3) controls whether or not supervariables are used. If ICNTL(3) = 0, supervariables are used and if ICNTL(3) = 1, variables are used. The default value is 0.

ICNTL(4) controls whether the user wishes to supply a global priority vector. If ICNTL(4) = 0, no priority vector is supplied and one is generated automatically using rooted level-set structures. If ICNTL(4) = 1, the user must supply a priority vector either in ORDER (DIRECT = .TRUE.) or in PERM (DIRECT = .FALSE.). The default value is 0.

ICNTL(5) controls the action taken if duplicate and/or out-of-range indices are detected in the element variable lists. If ICNTL(5)=0 and such indices are detected, they are removed, a warning is issued and the computation continues; if ICNTL(5)=1, the computation terminates. The default value is 0.

ICNTL(6) controls whether the user wishes to supply the weights for the priority function. If ICNTL(6)=0, no weights are supplied and the weights used then depend on the algorithm chosen by the user (see Section 4). If ICNTL(6) = 1, the user must supply weights in WT. The default value is 0.

ICNTL(7) to ICNTL(10) are currently not used but are given the default value 0.

2.1.2 To reorder the elements

The single precision version

```
      CALL MC63A(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,VARS,SVAR,
     +           WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)
```

The double precision version

```
      CALL MC63AD(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,VARS,SVAR,
     +           WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)
```

DIRECT is a LOGICAL variable that must be set by the user. DIRECT controls which reordering algorithm is implemented. If DIRECT = .TRUE., an direct element reordering algorithm is implemented; if DIRECT = .FALSE., an indirect element reordering algorithm is implemented.

N is an INTEGER variable that must be set by the user to the largest integer used to index a variable in the finite-element problem. Note that the variables need not be numbered contiguously and, in this case, N will be larger than *n*, the number of degrees of freedom. This argument is not altered by the routine. **Restriction:** N $\geq$ 1.

NELT is an INTEGER variable that must be set by the user to the total number of finite elements in the problem. This argument is not altered by the routine. **Restriction:** NELT $\geq$ 1.

NE is an INTEGER variable that must be set by the user to be at least as large as the total number of entries in the element variable lists. This argument is not altered by the routine. **Restriction:** NE $\geq$ 1.

ELTVAR is an INTEGER array of length NE. On entry, ELTVAR must contain lists of the variable indices belonging to each of the finite elements, with those for element 1 preceding those for element 2, and so on. If duplicate or variable indices outside the range [1, N] are detected, they are removed and the computation continues (ICNTL(5) = 0, the default) or the computation terminates with ELTVAR unchanged (ICNTL(5) = 1). On successful return, ELTVAR holds lists of supervariables (ICNTL(3) = 0) or variables (ICNTL(3) = 1) belonging to the elements.

ELTPTR is an INTEGER array of length NELT+1. On entry, ELTPTR(IELT) must contain the position in ELTVAR of the first variable in element IELT (IELT =1, 2,..., NELT), and ELTPTR(NELT+1) must be set to the position after the last variable in the last element. On return, ELTPTR holds corresponding data for the revised ELTVAR.

ORDER is an INTEGER array of length NELT. It need be set on entry only if DIRECT = .TRUE. **and** ICNTL(4) = 1. In this case, ORDER must hold positive global priority values for the elements (see Section 4, equation (2)). ORDER may be a permutation, in which case the element for which ORDER(IELT) = 1 is likely to be chosen first in the new assembly order and the element for which ORDER(IELT) = NELT is likely to be chosen last. On exit, the order in which the elements should be assembled is given by ORDER(1), ORDER(2),..., ORDER(NELT).

PERM is an INTEGER array that is only accessed if DIRECT = .FALSE. **and** ICNTL(4) = 1. In this case, PERM must be of length N and, if variable I is used to index a variable, PERM(I) must be set by the user to hold the positive global priority value for the variable I (see Section 4, equation (1)). PERM may be a permutation, in which case the variable for which PERM(I) = 1 is the variable with lowest priority and the variable for which PERM(I) = N is the variable with highest priority. This argument is not altered by the routine.

NSUP is an INTEGER variable that need not be set on entry. On successful return, if ICNTL(3) = 0 (the default), NSUP holds the number of supervariables. If ICNTL(3) = 1, on exit NSUP = N.

VARS is an INTEGER array of length N that need not be set on entry. On successful return, VARS(IS) holds the number of variables in supervariable IS, IS=1, 2,..., NSUP. If supervariables are not used, VARS(I) is set to 1 if I is used to index a variable and to 0 otherwise, I = 1, 2,..., N.

SVAR is an INTEGER array of length N that need not be set on entry. On successful return, SVAR(I) holds the supervariable to which variable I belongs, I = 1, 2,..., N. If variable I does not appear in the element lists, SVAR(I) = 0.

WT is a REAL (DOUBLE PRECISION in the D version) array of length 3. If ICNTL(6) = 1, WT must be set by the user to hold the weights that are used in the priority function that is minimized when the element assembly order is computed (see Section 4). Default values are used if ICNTL(6) = 0. WT(3) is not used if DIRECT = .FALSE. On return, WT holds the weights used in the priority function.

LIW is an INTEGER variable which defines the length of the work array IW. The workspace required depends upon whether the direct or the indirect element reordering algorithm is used, and upon whether supervariables are used (that is, the workspace depends upon the parameters DIRECT and ICNTL(3)). Lower bounds on the workspace needed are given by

LIW ≥ max(NE+3*NELT+NSUP+2, 2*N) if DIRECT = .TRUE. and ICNTL(3) = 0.

LIW ≥ 2*(NELT+N+1)+max(NE, 4*NELT) if DIRECT = .TRUE. and ICNTL(3) = 1.

LIW ≥ max(NE + NELT + 3*NSUP + 2, 2*N) if DIRECT = .FALSE. and ICNTL(3) = 0.

LIW ≥ 3*N+2+NELT+max(NE, 3*N) if DIRECT = .FALSE. and ICNTL(3) = 1.

Upper bounds on the workspace required are given by

LIW ≤ max(NE, 4*NELT)+2*max(NELT,N)+3+NELT* (maxel+1) if DIRECT = .TRUE., where maxel is the maximum number of elements to which any one variable belongs.

LIW ≤ max(NE, 3*N)+3*N+2+NELT*nodes*nodes if DIRECT = .FALSE., where nodes is the maximum number of variables in an element.

If LIW is too small, a value which will suffice is returned in INFO(5). This argument is not altered by the routine.

IW is an INTEGER array of length LIW. This array is used by the subroutine as workspace.

LW is an INTEGER variable which defines the length of the work array W. If DIRECT = .TRUE., LW must be at least NELT, and if DIRECT = .FALSE., LW must be at least NSUP. If LW is too small, a value which will suffice is returned in INFO(6). This argument is not altered by the routine.

W is a REAL (DOUBLE PRECISION in the D version) array of length LW. This array is used by the subroutine as workspace.

ICNTL is an INTEGER array of length 10 that must be set by the user to hold control parameters. Default values are set by a call to MC63I/ID. This argument is not altered by the routine.

INFO is an INTEGER array of dimension 15 that need not be set on entry.

INFO(1) is used as an error flag. On a successful exit, it is set to:

0 – Fully successful.

+1 – A warning has been issued. Further details are given in INFO(2), INFO(3), and RINFO.

If a fatal error has been detected, INFO(1) is set to a negative value:

–1 – N ≤ 0 or NELT ≤ 0 or NE < ELTPTR(NELT+1)-1. Immediate return with input parameters unchanged.

–2 – Failure due to insufficient space allocated to the array W. INFO(6) is set to a value that will suffice for LW.

–3 – ICNTL(5)=1 and duplicate or out-of-range indices detected in ELTVAR. The number of such indices is given by INFO(2) and INFO(3). This error is also returned if ICNTL(5)=0 and all the indices in an element are found to be out-of-range.

–4 – Failure due to insufficient space allocated to the array IW. INFO(5) is set to a value that will suffice for LIW.

INFO(2) holds the number of variable indices that were removed because they were duplicates.

INFO(3) holds the number of variable indices that were removed because they were out-of-range.

INFO(4) holds the number *n* of variables in the problem.

INFO(5) holds the amount of integer workspace used by the subroutine. If the user has provided insufficient integer workspace (INFO(1) = -4), INFO(5) is set to a value which will suffice for LIW (this value may be larger than the minimum workspace required).

INFO(6) holds the amount of real workspace used by the subroutine. If the user has provided insufficient real workspace (INFO(1) = -2), INFO(6) is set a value which will suffice for LW.

INFO(7) indicates whether the arrays ELTVAR and ELTPTR have been altered. If INFO(7) = 0, ELTVAR and ELTPTR are unchanged, otherwise they have been altered.

INFO(8) to INFO(15) are not currently used.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of dimension 6 that need not be set on entry. On successful exit, RINFO contains the following information.

RINFO(1) holds the maximum wavefront for the initial element order 1, 2,..., NELT.

RINFO(2) holds the r.m.s. wavefront for the initial element order 1, 2,..., NELT.

RINFO(3) holds the profile for the initial element order 1, 2,..., NELT.

RINFO(4) holds the maximum wavefront for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

RINFO(5) holds the r.m.s. wavefront for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

RINFO(6) holds the profile for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

2.1.3 To compute statistics for a given element assembly order

The single precision version

```
CALL MC63B(JCNTL,N,NSUP,NELT,NE,ELTVAR,ELTPTR,VARS,ORDER,IW,INFO,RINFO)
```

The double precision version

```
CALL MC63BD(JCNTL,N,NSUP,NELT,NE,ELTVAR,ELTPTR,VARS,ORDER,IW,INFO,RINFO)
```

JCNTL is an INTEGER variable that controls whether the lists of variable indices are checked for errors.

0 – No checks made.

+1 – Any duplicate or out-of-range indices are removed and the computation continues.

–1 – The computation terminates if any duplicate or out-of-range indices are found.

If JCNTL = +1 or -1, INFO(2) and INFO(3) provide information on the number of duplicate and out-of-range indices. This argument is not altered by the routine.

N is an INTEGER variable that must be set by the user to the largest integer used to index a variable in the finite-element problem. This argument is not altered by the routine.

NSUP is an INTEGER variable. If MC63A/AD has been called, it should be unchanged since the call to MC63A/AD. Otherwise, NSUP should be set to N, the largest integer used to index a variable in the finite-element problem. This argument is not altered by the routine.

NELT is an INTEGER variable that must be set by the user to the total number of finite elements in the problem. This argument is not altered by the routine.

NE is an INTEGER variable that must be set by the user to be at least as large as the total number of entries in the element variable lists. This argument is not altered by the routine.

ELTVAR is an INTEGER array of length NE. It may be as returned by MC63A/AD. Alternatively, it may be set by the user so that ELTVAR contains lists of the variable indices belonging to each of the finite elements, with those for element 1 preceding those for element 2, and so on. This argument is unchanged on exit unless JCNTL = 1 and duplicate and/or out-of-range indices are detected.

ELTPTR is an INTEGER array of length NELT+1. It may be as returned by MC63A/AD. Alternatively, it may be set by the user so that ELTPTR(I) contains the position in ELTVAR of the first variable in element I (I=1, 2,..., NELT), and ELTPTR(NELT+1) must be set to the position after the last variable in the last element. On exit, ELTPTR holds corresponding data for the revised ELTVAR.

VARS is an INTEGER array of length NSUP. If MC63A/AD was called, it should be unchanged since the call to MC63A/AD and it is not altered by the subroutine. Otherwise, (the case NSUP = N) VARS need not be set by the user.

ORDER is an INTEGER array of length NELT which must be set by the user so that the order in which the elements are assembled is given by ORDER(1), ORDER(2),..., ORDER(NELT). This argument is not altered by the routine.

IW is an INTEGER array of length NSUP. This array is used by the subroutine as workspace.

INFO is a REAL (DOUBLE PRECISION in the D version) array of length 4.

INFO(1) is used as an error flag. On exit, it is set to:

0 – No duplicate or out-of-range indices detected.

+1 – JCNTL = 1 and some duplicate or out-of-range indices have been removed.

–1 – JCNTL = -1 and duplicate or out-of-range indices detected.

Further details are given in INFO(2) and INFO(3).

INFO(2) holds the number of duplicate indices (only set if JCNTL = 1).

INFO(3) holds the number of out-of-range indices (only set if JCNTL = 1).

INFO(4) holds the number *n* of variables in the problem.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of length 3. On exit, RINFO contains the following information.

RINFO(1) holds the maximum wavefront for the given element order.

RINFO(2) holds the r.m.s. wavefront for the given element order.

RINFO(3) holds the profile for the given element order.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: Subroutines internal to the package are MC63C/CD, MC63D/DD, MC63E/ED, MC63F/FD, MC63G/GD. In addition, MC60C/CD, MC60H/HD, MC60L/LD, MC60O/OD are called.

Input/output: Error messages on unit ICNTL(1) (ICNTL(1) ≤ 0 suppresses them) and warnings on unit ICNTL(2) (ICNTL(2) ≤ 0 suppresses them).

Restrictions:

N ≥ 1, NELT ≥ 1, NE ≥ 1.

4 METHOD

MC63A/AD accepts lists of variables belonging to the elements and, after performing initial checks on the user's data, calls MC63B/BD to compute statistics for the natural element order 1,2,..,*nelt*. MC63B/BD also checks the element variable lists for out-of-range and duplicate indices. Any such entries are removed and the computation either continues after issuing a warning message or terminates if this has been requested.

If supervariables are wanted (ICNTL(3) = 0), they are constructed using subroutine MC60O/OD from the HSL profile reduction package MC60. Otherwise, each variable is treated as a supervariable. The element variable lists are overwritten by element supervariable lists. A map of variable to supervariable indices allows the user to later restore the element variable lists, if desired.

For each supervariable, the number of elements involving it is counted. Lists of the elements associated with the supervariables are then constructed. If the user has selected the direct element reordering algorithm (DIRECT = .TRUE), the element connectivity graph is constructed from the supervariable lists, otherwise the supervariable connectivity graph is constructed from the element lists.

Both the direct and indirect algorithms are based upon the modifications of Sloan's algorithm (Sloan 1986) first introduced by Duff, Reid, and Scott (1989).

In the indirect element reordering algorithm (DIRECT = .FALSE), MC60C/CD is used to reorder the supervariables. The priority function that is minimized when choosing the next supervariable in the order is

$$\mathtt{WT(1)}\, deg(s) + \mathtt{WT(2)}\, vglob(s) \qquad (1)$$

where *deg*(*s*) is the number of variables that will enter the front if supervariable *s* is chosen next, v is a normalizing factor, and *glob*(*s*) is the (positive) global priority value of supervariable *s* (generated automatically or provided in PERM). On the basis of our numerical experiments, the default values for the weights WT(1), WT(2) are (2,1) if the global priority vector is based on a rooted level-set structure (ICNTL(4) = 0) and (1,2) for a global priority vector based on the spectral method. Full details are given in Reid and Scott (1998). Once the supervariables have been reordered, the elements are resequenced in ascending order of their earliest supervariable in the new supervariable order. The new supervariable indices are not preserved.

In the direct element ordering algorithm, the element connectivity graph is relabeled using the priority function

$$\mathtt{WT(1)}\ ngain(ielt) + \mathtt{WT(2)}\ v\, glob(ielt) + \mathtt{WT(3)}\ nadj(ielt) \tag{2}$$

where *ngain*(*ielt*) is the number of variables element *ielt* will introduce into the front less the number that can then be eliminated, *nadj*(*ielt*) is the number of elements adjacent to element *ielt* that have not yet been relabeled, v is a normalizing factor, and *glob*(*ielt*) is the (positive) global priority value of element *ielt* (generated automatically or provided in ORDER). At each stage, from a list of eligible elements, the element that minimizes the priority function is chosen to be next in the element assembly order. The default values for the weights are (10,5,1) if the global priority vector is based on a rooted level-set structure (ICNTL(4) = 0) and (1,2,0) for a global priority vector based on the spectral method. For full details the user is referred to Scott (1998).

A final call to MC63B/BD (without error checking) computes statistics for the new element order.

References

Duff, I. S., Reid, J. K., and Scott, J. A. (1989). The use of profile reduction algorithms with a frontal code. *Int. J. Numer. Meth. Engng.* **28**, 2555-2568.

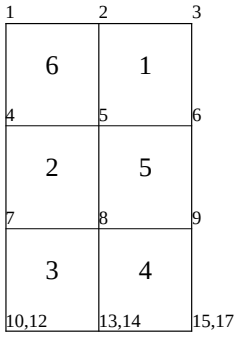
Reid, J. K. and Scott, J. A. (1998). Ordering symmetric sparse matrices for small profile and wavefront. Technical Report RAL-TR-98-016, Rutherford Appleton Laboratory.

Scott, J. A. (1998). On ordering elements for a frontal solver. Technical Report RAL-TR-98-031, Rutherford Appleton Laboratory.

Sloan, S. W. (1986). An algorithm for profile and wavefront reduction of sparse matrices. Inter. J. Numer. Meth. Engng. **23**, 239-251.

5 EXAMPLE OF USE

The following program provides an example of the use of MC63. We wish to reorder the elements in the following simple finite-element mesh comprising six 4-noded quadrilateral elements. The elements are initially numbered arbitrarily. Note that the variables are not numbered contiguously and at some nodes there is more than one freedom.




```
C  Example to illustrate the use of MC63A.
C  Both the direct and the indirect element reordering algorithms
C  are employed.
```

```
C     .. Parameters ..
      INTEGER MELT,MXN,MZ,LIW
      PARAMETER (MELT=6,MXN=20,MZ=30,LIW=200)
C     ..
C     .. Local Scalars ..
      INTEGER I,IDUM,LW,N,NE,NELT,NSUP
      LOGICAL DIRECT
C     ..
C     .. Local Arrays ..
      REAL WT(3),RINFO(6),W(MXN)
      INTEGER CPTR(MELT+1),CVAR(MZ),ELTPTR(MELT+1),ELTVAR(MZ),
     +        ICNTL(10),INFO(15),IW(LIW),
     +        ORDER(MELT),PERM(1),SVAR(MXN),VARS(MXN)
C     ..
C     .. External Subroutines ..
      EXTERNAL MC63A,MC63I
C     ..
C   Read in the finite-element data
      READ (5,*) N,NELT
      READ (5,*) (ELTPTR(I),I=1,NELT+1)
      NE = ELTPTR(NELT+1) - 1
      READ (5,*) (ELTVAR(I),I=1,NE)
      LW = MXN

C Take a copy of ELTVAR, ELTPTR as altered by MC63A/AD
C and we want to run direct and indirect algorithms
      DO 5 I = 1,NELT+1
         CPTR(I) = ELTPTR(I)
    5 CONTINUE
      DO 10 I = 1,NE
         CVAR(I) = ELTVAR(I)
   10 CONTINUE

      CALL MC63I(ICNTL)

      DO 20 IDUM = 1,2

         IF (IDUM.EQ.1) THEN
            DIRECT = .TRUE.
            WRITE (6,'(/3X,A)') '*** Direct element reordering ***'
         ELSE
            DIRECT = .FALSE.
            WRITE (6,'(/3X,A)') '*** Indirect element reordering ***'
C Reset ELTVAR, ELTPTR
            DO 15 I = 1,NELT+1
               ELTPTR(I) = CPTR(I)
   15       CONTINUE
            DO 16 I = 1,NE
               ELTVAR(I) = CVAR(I)
   16       CONTINUE
         END IF

         CALL MC63A(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,
     +              VARS,SVAR,WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)

C  Check for errors
         IF (INFO(1).LT.0) GO TO 30

         WRITE (6,'(A,I12,I12/A,1P,D12.4,1P,D12.4/
     +       A,1P,D12.4,1P,D12.4)')
```

```
     +     ' Original/new max. wavefront             = ',
     +        INT(RINFO(1)),INT(RINFO(4)),
     +     ' Original/new r.m.s. wavefront           = ',
     +        RINFO(2),RINFO(5),
     +     ' Original/new profile                    = ',
     +        RINFO(3),RINFO(6)
         WRITE (6,'(A,I12/A,I12,I12)')
     +     ' Workspace used                          = ',
     +        INFO(5),
     +     ' Number of variables/supervariables      = ',
     +        INFO(4),NSUP
         WRITE (6,'(/3X,A/6I5)') 'The new element order is :',
     +     (ORDER(I),I=1,NELT)

   20 CONTINUE
      GO TO 40

   30 WRITE (6,*) ' Unexpected error return'

   40 STOP
      END
```

The input data used for this problem is:

```
   17   6
    1   5   9  15  21  25  29
    2   5   3   6   4   5   7   8   7   8  10  12  4  13
    8  13   9  14  17  15   5   8   9   6   1   2   5   4
```

This produces the following output:

```
     *** Direct element reordering ***
 Original/new max. wavefront             =           10           7
 Original/new r.m.s. wavefront           =   6.3823D+00  4.6476D+00
 Original/new profile                    =   8.7000D+01  6.6000D+01
 Workspace used                          =           86
 Number of variables/supervariables      =           15          13

     The new element order is :
      1    6    5    2    3    4

     *** Indirect element reordering ***
 Original/new max. wavefront             =           10           7
 Original/new r.m.s. wavefront           =   6.3823D+00  4.6476D+00
 Original/new profile                    =   8.7000D+01  6.6000D+01
 Workspace used                          =          142
 Number of variables/supervariables      =           15          13

     The new element order is :
      1    6    5    2    3    4
```