MC60

1 SUMMARY

This subroutine uses a variant of Sloan's method to calculate a symmetric permutation that aims to **reduce the profile and wavefront of a sparse $n \times n$ matrix A with a symmetric sparsity pattern.** Alternatively, the Reverse Cuthill-McKee Method may be requested to reduce the bandwidth. There are optional facilities for looking for sets of columns with identical patterns and taking advantage of them. There is also an option for computing a row order that would be appropriate for use with a row-by-row frontal solver (for example, the equation entry to `MA42`). These optional facilities may also be used independently.

Definitions of the matrix profile, wavefront etc. are included in Section 4.

ATTRIBUTES — Version: 1.1.0 (26 October 2012). **Types:** Real (single, double). **Calls:** None. **Original date:** January 1998. **Origin:** J. K. Reid and J. A. Scott (Rutherford Appleton Laboratory).

2 HOW TO USE THE PACKAGE

2.1 Argument lists

There are seven entries to this package:

`MC60A` accepts the pattern of the lower-triangular part of the matrix and constructs the pattern of the whole matrix. There are extensive checks on the data. This is the usual initial entry.

`MC60B` looks for sets of columns with identical patterns. We refer to the set of variables that correspond to a set of identical columns as a supervariable. A permutation is constructed that places the variables of each supervariable together. The pattern is replaced by that of the permuted matrix represented as supervariables (that is, by its condensed equivalent).

`MC60C` chooses a supervariable permutation that aims to reduce the profile and wavefront or the bandwidth. It also provides pseudoperipheral pairs of nodes of the components of the supervariable graph of the matrix.

`MC60D` constructs the permutation for the variables that corresponds to a given permutation for the supervariables.

`MC60E` uses a given permutation for the supervariables to construct the corresponding row order, as required by a row-by row frontal solver, such as `MA42`.

`MC60F` uses a given permutation for the supervariables to compute the profile, the maximum wavefront, the semibandwidth, and the root-mean-square wavefront for the permuted matrix.

`MC60G` uses a given row order to compute the maximum row and column front sizes, the root-mean-square row front size and mean frontal matrix size for a row-by-row frontal method.

Only the initial entry provides extensive checks on the data. If it is known that there are few supervariables, `MC60B` will not be needed. On the other hand, `MC60B` may be used in combination with another algorithm for choosing a permutation.

2.1.1 Initial entry

The single precision version

```
   CALL MC60A(N,LIRN,IRN,ICPTR,ICNTL,IW,INFO)
```

The double precision version

```
   CALL MC60AD(N,LIRN,IRN,ICPTR,ICNTL,IW,INFO)
```

N is an INTEGER variable that must be set by the user to the order of the matrix. N is not altered. **Restriction:** N $\geq$ 1.

LIRN is an INTEGER variable that must be set by the user to the length of the array IRN, which must be large enough to hold the pattern of the whole matrix (excluding duplicated and out-of-range indices). 2*(ICPTR(N+1)-1) is always sufficient. LIRN is not altered.

IRN is an INTEGER array of length LIRN whose leading part must be set by the user to hold the row indices of the entries in the lower triangle (including those on the diagonal) of the matrix **A**. The entries of each column must be contiguous. The entries of column J must precede those of column J+1, J $= 1,2,\ldots,$N-1, and there must be no wasted space between the columns. Row indices within a column may be in any order. On successful return, the array will be changed to hold the row indices of the whole matrix **A** in the same format.

ICPTR is an INTEGER array of length N+1 that must be set by the user so that ICPTR(J) points to the position in the array IRN of the first entry in column J, J = 1, 2, ..., N, and ICPTR(N+1)-1 must be the position of the last entry. On successful return, ICPTR holds corresponding data for the revised IRN.

ICNTL is an INTEGER array of length 2 that controls the action:

ICNTL(1) controls whether the computation terminates if duplicated or out-of-range indices are detected:

0 Terminates if any duplicated or out-of-range indices found.

1 Any duplicated or out-of-range indices are ignored.

ICNTL(2) controls printing of diagnostic messages:

0 No diagnostic messages are required.

$>$0 The unit number for diagnostic messages.

ICNTL is not altered.

IW is an INTEGER array of length N that is used by the subroutine as workspace.

INFO is an INTEGER array of length 4 that need not be set by the user.

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

0 No out-of-range or duplicated indices.

1 Some out-of-range or duplicated indices when ICNTL(1) = 1.

If a fatal error has been detected, it is set to a negative value:

-1 N $<$ 1 or LIRN less than ICPTR(N+1)-1. Immediate return with IRN and ICPTR unchanged.

-2 LIRN is too small. INFO(4) is set to the minimum value that will suffice. If ICNTL(1) = 1, any out-of-range or duplicated variable indices will have been excluded from IRN and ICPTR. Otherwise, IRN and ICPTR are unchanged.

-3 ICNTL(1) = 0 and one or more variable indices either lies outside the lower triangle of the matrix or is duplicated (see INFO(2) and INFO(3)). IRN and ICPTR are unchanged.

INFO(2) holds the number of variable indices in IRN found to be out-of-range.

INFO(3) holds the number of indices in IRN that represent duplicates of previous entries.

INFO(4) holds the minimum value that will suffice for LIRN, unless INFO(1) = -1.

2.1.2 To find supervariables and compress the pattern

The single precision version

```
  CALL MC60B(N,LIRN,IRN,ICPTR,NSUP,SVAR,VARS,IW)
```

The double precision version

```
  CALL MC60BD(N,LIRN,IRN,ICPTR,NSUP,SVAR,VARS,IW)
```

N is an INTEGER variable that must be set by the user to the order of the matrix. N is not altered.

LIRN is an INTEGER variable that must be set by the user to the length of the array IRN. LIRN is not altered.

IRN is an INTEGER array of length LIRN. On entry, it may be as returned by MC60A/AD. Alternatively, it may be set by the user to hold the row indices of the whole matrix. The entries of each column must be contiguous. The entries of column J must precede those of column J+1, J = 1, 2, ..., N-1, and there must be no wasted space between the columns. The row indices within a column may be in any order. Columns with no entries are permitted. No checks on the format are performed. On successful return, IRN holds the row indices of the entries in the condensed matrix, using the same format.

ICPTR is an INTEGER array of length N+1. On entry, it may be as returned by MC60A/AD. Alternatively, it may be set by the user so that ICPTR(J) points to the position in the array IRN of the first entry in column J, J = 1, 2, ..., N, and ICPTR(N+1)-1 points to the last entry. On successful return, ICPTR holds corresponding data for the condensed matrix.

NSUP is an INTEGER variable that need not be set on entry. On return, it holds the number of supervariables.

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.

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.

IW is an INTEGER array of length 2*N+2 that is used by the subroutine as workspace.

2.1.3 To find supervariable permutation

A normal call to MC60C/CD will follow a successful call to MC60B/BD, in which case the arguments N, NSUP, LIRN, IRN, ICPTR, and VARS should be unchanged since return from MC60B/BD. However, it may be called independently, so we describe these arguments as if they were provided afresh.

By making each supervariable consist of a single variable, MC60C/CD may also be used to find a permutation for the variables. In this case, NSUP must equal N and all elements of VARS must equal 1; N, LIRN, IRN, ICPTR will normally be unchanged since return from MC60A/AD, but they may be provided afresh. We do not recommend this option unless it is known that the problem has few supervariables.

The single precision version

```
CALL MC60C(N,NSUP,LIRN,IRN,ICPTR,VARS,JCNTL,PERMSV,WEIGHT,PAIR,INFO,IW,W)
```

The double precision version

```
CALL MC60CD(N,NSUP,LIRN,IRN,ICPTR,VARS,JCNTL,PERMSV,WEIGHT,PAIR,INFO,IW,W)
```

N is an INTEGER variable that must be set by the user to the order of the matrix. This argument is not altered.

NSUP is an INTEGER variable that must be set to hold the number of supervariables. NSUP is not altered.

LIRN is an INTEGER variable that must be set by the user to the length of the array IRN. LIRN is not altered.

IRN is an INTEGER array of length LIRN. It must be set by the user to hold the row indices of the condensed matrix. The entries of each column must be contiguous. The entries of column J must precede those of column J+1, J = 1, 2, ..., NSUP-1, and there must be no wasted space between the columns. The row indices within a column may be in any order. No checks on the format are performed. IRN is not altered.

ICPTR is an INTEGER array of length NSUP+1. It must be set by the user so that ICPTR(J) points to the position in the array IRN of the first entry in column J, J = 1, 2, ..., NSUP, and ICPTR(NSUP+1)-1 points to the last entry. ICPTR is not altered.

VARS is an INTEGER array of length NSUP. VARS(IS) must hold the number of variables in supervariable IS, IS = 1, 2, ..., NSUP. VARS is not altered.

JCNTL is an INTEGER array of length 2 that controls the action:

JCNTL(1) controls the choice of algorithm:

0 Sloan's algorithm for reducing profile and wavefront.
1 Reverse Cuthill-McKee algorithm (RCM) for reducing bandwidth.

JCNTL(2) controls algorithmic details:

0 Automatic choice of pseudoperipheral pairs.
1 Pseudoperipheral pairs specified in PAIR.
2 Global priority vector given in PERMSV (Sloan's algorithm only).

JCNTL is not altered.

PERMSV is an INTEGER array of length NSUP. It need be set on entry only if JCNTL(2)=2, and in this case must hold positive global priority values for the supervariables; this may be a permutation, in which case the supervariable for which PERMSV(IS) = 1 is likely to be chosen first and the supervariable for which PERMSV(IS) = NSUP is likely to be chosen last. On exit, the position of supervariable IS in the new ordering is given in all cases by PERMSV(IS), IS = 1, 2, ..., NSUP.

WEIGHT is a REAL (DOUBLE PRECISION in the D version) array of length 2. For Sloan's algorithm (JCNTL(1)=0), it must be set to the weights W_1 and W_2 in the priority function that is minimized when choosing the next supervariable in the order. The value of the function is

$$W_1 \deg(s) + W_2 \nu \, \mathrm{glob}(s)$$

where deg(s) is the number of variables that will enter the front if supervariable s is chosen next, ν is a normalizing factor (see Section 4.3), and glob(s) is the (positive) global priority value of supervariable s (generated automatically or provided in PERMSV). The choice of weights is discussed in Section 4.3.

PAIR is an INTEGER array of shape (2,NSUP/2). If JCNTL(2)=0, it need not set on entry and on return PAIR(1,IC), PAIR(2,IC) hold the pseudoperipheral pair for nontrivial component IC, IC = 1, 2, ..., INFO(1). The first component is the largest. If JCNTL(2)=1, it must be set on entry to the pseudoperipheral pairs of the components and is not altered; the first component need not be the largest. If JCNTL(2) = 2, it is not used.

INFO is an INTEGER array of length 4 that need not be set by the user. On exit,

INFO(1)	holds the number of nontrivial components (two or more nodes) in the graph of the condensed matrix,
INFO(2)	holds the number of variables in the largest component of the graph,
INFO(3)	holds the number of level sets in the level-set structure of the largest component, and
INFO(4)	holds the width of the level-set structure of the largest component.

IW is an INTEGER array of length 3*NSUP+1 that is used by the subroutine as workspace.

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

2.1.4 To find permutation for variables from supervariable permutation

The single precision version

```
CALL MC60D(N,NSUP,SVAR,VARS,PERMSV,PERM,POSSV)
```

The double precision version

```
CALL MC60DD(N,NSUP,SVAR,VARS,PERMSV,PERM,POSSV)
```

N is an INTEGER variable that must hold the order of the matrix. N is not altered.

NSUP is an INTEGER variable that must hold the number of supervariables. NSUP is not altered.

SVAR is an INTEGER array of length N. SVAR(I) must hold the supervariable to which variable I belongs, I = 1, 2, ..., N. SVAR is not altered.

VARS is an INTEGER array of length NSUP. VARS(IS) must hold the number of variables in supervariable IS, IS = 1, 2, ..., NSUP. VARS is not altered.

PERMSV is an INTEGER array of length NSUP. It may be as returned by MC60C/CD. Alternatively, it may be set by the user so that PERMSV(IS) holds the position to which supervariable IS is permuted, IS = 1, 2, ..., NSUP. PERMSV is not altered..

PERM is an INTEGER array of length N that need not be set by the user. On return, PERM(I) holds the position of variable I, I = 1, 2, ..., N, in the permuted list of variables.

POSSV is an INTEGER array of length NSUP that need not be set by the user. On return, POSSV(IS) holds the position of the first variable of supervariable IS, IS = 1, 2, ..., NSUP, in the permuted list of variables.

2.1.5 To find a row-by-row frontal order from a supervariable permutation

The single precision version

```
CALL MC60E(N,NSUP,LIRN,IRN,ICPTR,SVAR,VARS,PERMSV,PERM,IW)
```

The double precision version

```
CALL MC60ED(N,NSUP,LIRN,IRN,ICPTR,SVAR,VARS,PERMSV,PERM,IW)
```

N is an INTEGER variable that must hold the order of the matrix. N is not altered.

NSUP is an INTEGER variable that must hold the number of supervariables. NSUP is not altered.

LIRN is an INTEGER variable that must be set by the user to the length of the array IRN. LIRN is not altered.

IRN is an INTEGER array of length LIRN. It must be as for MC60C/CD. No checks on the format are performed. IRN is not altered.

ICPTR is an INTEGER array of length NSUP+1. It must be as for MC60C/CD. No checks on the format are performed. ICPTR is not altered.

SVAR is an INTEGER array of length N. SVAR(I) must hold the supervariable to which variable I belongs, I = 1, 2, ..., N. SVAR is not altered.

VARS is an INTEGER array of length NSUP. VARS(IS) must hold the number of variables in supervariable IS, IS = 1, 2, ..., NSUP. VARS is not altered.

PERMSV is an INTEGER array of length NSUP. It may be as returned by MC60C/CD. Alternatively, it may be set by the user so that PERMSV(IS) holds the new index for supervariable IS, IS = 1, 2, ..., NSUP. On return, the row-by-row order for the rows of the condensed matrix is PERMSV(1), PERMSV(2), ..., PERMSV(NSUP).

PERM is an INTEGER array of length N that need not be set by the user. On return, the row-by-row order for the rows of the matrix **A** is PERM(1), PERM(2), ..., PERM(N).

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

2.1.6 To compute the profile and wavefront for a supervariable permutation

The single precision version

```
CALL MC60F(N,NSUP,LIRN,IRN,ICPTR,VARS,PERMSV,IW,RINFO)
```

The double precision version

```
CALL MC60FD(N,NSUP,LIRN,IRN,ICPTR,VARS,PERMSV,IW,RINFO)
```

N is an INTEGER variable that must hold the order of the matrix. N is not altered.

NSUP is an INTEGER variable that must hold the number of supervariables. NSUP is not altered.

LIRN is an INTEGER variable that must be set by the user to the length of the array IRN. LIRN is not altered.

IRN is an INTEGER array of length LIRN. It must be as for MC60C/CD. No checks on the format are performed. IRN is not altered.

ICPTR is an INTEGER array of length NSUP+1. It must be as for MC60C/CD. No checks on the format are performed. ICPTR is not altered.

VARS is an INTEGER array of length NSUP. VARS(IS) must hold the number of variables in supervariable IS, IS = 1, 2, ..., NSUP. VARS is not altered.

`PERMSV` is an `INTEGER` array of length `NSUP`. It may be as returned by `MC60C/CD`. Alternatively, it may be set by the user so that `PERMSV(IS)` holds the new index for supervariable `IS`, `IS` = `1`, `2`, ..., `NSUP`. If data for the original order are required, `PERMSV(IS)` should be set to `IS`, `IS` = `1`, `2`, ..., `NSUP`. `PERMSV` is not altered.

`IW` is an `INTEGER` array of length `2*NSUP+1` that is used by the subroutine as workspace.

`RINFO` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `4` that need not be set by the user. On exit `RINFO(1)`, `RINFO(2)`, `RINFO(3)`, and `RINFO(4)` hold, respectively, the profile, the maximum wavefront, the semibandwidth, and the root-mean-square wavefront for the permuted matrix.

2.1.7 To compute the front sizes for a row-by-row frontal method

The single precision version

```
CALL MC60G(N,NSUP,LIRN,IRN,ICPTR,VARS,PERMSV,IW,RINFO)
```

The double precision version

```
CALL MC60GD(N,NSUP,LIRN,IRN,ICPTR,VARS,PERMSV,IW,RINFO)
```

`N` is an `INTEGER` variable that must hold the order of the matrix. `N` is not altered.

`NSUP` is an `INTEGER` variable that must hold the number of supervariables. `NSUP` is not altered.

`LIRN` is an `INTEGER` variable that must be set by the user to the length of the array `IRN`. `LIRN` is not altered.

`IRN` is an `INTEGER` array of length `LIRN`. It must be as for `MC60C/CD`. No checks on the format are performed. `IRN` is not altered.

`ICPTR` is an `INTEGER` array of length `NSUP+1`. It must be as for `MC60C/CD`. No checks on the format are performed. `ICPTR` is not altered.

`VARS` is an `INTEGER` array of length `NSUP`. `VARS(IS)` must hold the number of variables in supervariable `IS`, `IS` = `1`, `2`, ..., `NSUP`. `VARS` is not altered.

`PERMSV` is an `INTEGER` array of length `NSUP`. It may be as returned by `MC60E/ED`. Alternatively, it may be set by the user so that the row-by-row order for the condensed matrix is `PERMSV(1)`, `PERMSV(2)`, ..., `PERMSV(NSUP)`. `PERMSV` is not altered.

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

`RINFO` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `4` that need not be set by the user. On exit `RINFO(1)`, `RINFO(2)`, `RINFO(3)`, and `RINFO(4)` hold, respectively, the maximum row and column front sizes, the root-mean-square row front size and the mean frontal matrix size.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: The subroutines documented here call the following subroutines of the `MC60` package: `MC60H/HD`, `MC60J/JD`, `MC60L/LD`, `MC60O/OD`, and `MC60P/PD`.

Input/output: If `ICNTL(2)` $>$ `0`, diagnostic messages on unit `ICNTL(2)` (`MC60A/AD` only).

Restrictions: `N` $\geq$ `1`.

4 METHOD

We first introduce some definitions. When defining the profile, wavefront and bandwidth, any zeros on the diagonal are regarded as nonzero. Let m_i denote the column index of the first nonzero in row i ($m_i \leq i$). The length of row i is defined to be $i - m_i + 1$ and the profile P of $\mathbf{A}$ is the sum of the lengths of the rows, that is,

$$P = \sum_{i=1}^{n} (i - m_i + 1).$$

The ith wavefront f_i of $\mathbf{A}$ is the number of rows that have nonzeros in the submatrix $\mathbf{A}(i:n, 1:i)$. It can be shown that

$$P = \sum_{i=1}^{n} f_i.$$

The maximum and mean-square wavefronts are, respectively,

$$\max_{1 \leq i \leq n} f_i \qquad \text{and} \qquad \frac{1}{n} \sum_{i=1}^{n} f_i^2.$$

$\mathbf{A}$ has bandwidth $2m+1$ and semibandwidth m if m is the smallest integer such that $a_{ij} = 0$ whenever $|i-j| > m$.

For the row-by-row frontal method, if $frow_i$ denotes the number of rows in the front at the ith elimination, the maximum row front size is defined as

$$\max_{1 \leq i \leq n} frow_i.$$

The maximum column front size is defined similarly, with $fcol_i$ the number of columns in the front at the ith elimination.
The mean-square row front size and mean frontal matrix size are, respectively, defined to be

$$\frac{1}{n} \sum_{i=1}^{n} frow_i^2 \qquad \text{and} \qquad \frac{1}{n} \sum_{i=1}^{n} (frow_i * fcol_i).$$

4.1 MC60A

For economy of storage, `MC60A` performs its work in place. A first pass looks for any out-of-range or repeated indices and removes them, or terminates if this has been requested. A second pass counts the number of entries that need to be added to each row to include the upper triangle. A third pass works through the rows in reverse order, moving them back to allow space for the additional entries. A final pass inserts the additional entries. There are extensive checks on the data. If the user already has the pattern of the whole matrix and does not wish to checks to made on the data, `MC60A` is not needed.

4.2 MC60B

`MC60B` constructs supervariables in $O(n+\tau)$ time, where n is the order of the matrix and τ is the number of entries, by working progressively so that after j steps we have the supervariable structure for the submatrix of the first j columns. We start with all variables in one supervariable (for the submatrix with no columns), then split it into two according to which rows do or do not have an entry in column 1, then split these according to the entries in column 2, etc. The splitting is done by moving the variables one at a time to the new supervariable. Further details are given by Reid and Scott (1998).

Note that this strategy requires the user to provide the indices of the entries on the diagonal since these affect whether the structures of columns are identical. This contrasts with `MC40`, which assumes that the diagonal entries are all nonzero.

The use of `MC60B` is optional. If it is known that there are few supervariables, `MC60B` will not be needed. On the other hand, `MC60B` may be used in combination with another algorithm for choosing an ordering.

4.3 MC60C

`MC60C` controls the main part of the algorithm. It works with the supervariable graph, which has *nsup* nodes and an edge between nodes i and j if entry i, j is present in the condensed matrix. It allows for the matrix being reducible (a permutation of a block diagonal matrix). In this case, each diagonal block of the permuted matrix will correspond to a component of the graph (set of nodes with no connections to other nodes). It orders any trivial components first by choosing any nodes that have no connections to other nodes. It then orders each nontrivial component in turn by calling other subroutines, which allows these other subroutines to work with a single component.

If `JCNTL(2)=0`, a pair of well-separated nodes (a pseudoperipheral pair) is selected for each nontrivial component by using a procedure which is a modification of that given by Gibbs, Poole, and Stockmeyer (1976). Further details are given by Reid and Scott (1998). Alternatively, the pairs may be specified by the user (`JCNTL(2)=1`).

Given a pseudoperipheral node, a *level-set structure rooted on the node* consists of the node itself at level 1 and at each level i the nodes that are neighbours of nodes at level $i-1$ but are not members of levels 1, 2, ..., $i-1$. The depth is the number of level sets and the width is the greatest number of variables associated with all the nodes of a single level. Once we have a pseudoperipheral pair, we chose the node whose rooted level set has the greater depth or, if they have the same depth, the lesser width. The ordering associated with this node by taking its last level set first, then its penultimate level set, etc. is used directly for the RCM method. The corresponding level-set indices are used as a global priority vector for Sloan's algorithm. The global priority vector may also be supplied by the user, for example, from a spectral ordering (see Barnard, Pothen, and Simon 1995), in which case we normalize it with the factor ν, chosen to make the range the same as if the level-set indices were in use.

In the method of Sloan (1986), each successive node is chosen to minimize a weighted average of the global priority function and the number of variables that will enter the front if this node is chosen next. For the weights, Sloan recommends the pair (2,1) if the global priority vector is based on a rooted level-set structure. For global priority vectors based on the spectral method, we found that (1,2) was often to be preferred. However the global priority vector was generated, we have found that on some problems the weights (16,1) are significantly better. By default, our driver `MC61` therefore calls `MC60C` twice, either with weights (2,1) and (16,1) or with weights (1,2) and (16,1), and takes the better result. It also checks that there is an improvement over the natural order.

Further details about the `MC60C` algorithms and their performance are given by Reid and Scott (1998).

4.4 MC60D, MC60E, MC60F, and MC60G

The remaining subroutines perform straightforward tasks of converting a supervariable ordering to an ordering for variables or rows, or providing statistics.

References

Barnard, S. T., Pothen, A., and Simon, H. (1995). A spectral algorithm for envelope reduction of sparse matrices. Numerical Linear Algebra with Applications **2**, 317-334.

Gibbs, N.E., Poole, W.G., and Stockmeyer, P.K. (1976). An algorithm for reducing the profile and bandwidth of a sparse matrix. SIAM J. Numer. Anal. **13**, 236-250.

Reid, J.K. and J.A. Scott. (1998). Ordering symmetric sparse matrices for small profile and wavefront. Technical Report RAL-TR-98-016, 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 a simple example of the use of `MC60`. It works with or without looking for supervariables.

```
      INTEGER LIRN,MAXN
      PARAMETER (LIRN=20, MAXN=5)
      INTEGER CASE,N,NNZ,IRN(LIRN),ICPTR(MAXN+1),ICNTL(2),
     *      IW(3*MAXN+1),INFO(4),NSUP,I,SVAR(MAXN),VARS(MAXN),
     *      PERM(MAXN),PERMSV(MAXN),JCNTL(2),PAIR(2,MAXN/2)
      REAL WEIGHT(2),W(MAXN),RINFO(4)
      CHARACTER ALG

C Set parameter values
      ICNTL(1) = 0
      ICNTL(2) = 6
      JCNTL(1) = 0
      JCNTL(2) = 0
      WEIGHT(1) = 2.0
      WEIGHT(2) = 1.0

      DO 20 CASE = 1,2
C Read in data for the lower-triangular part
        READ (5,*) N,NNZ
        READ (5,*) (IRN(I),I = 1,NNZ)
        READ (5,*) (ICPTR(I),I = 1,N+1)

C Construct pattern of whole matrix
        CALL MC60A(N,LIRN,IRN,ICPTR,ICNTL,IW,INFO)

C Check for an error return
        IF (INFO(1).NE.0) THEN
          WRITE(6,'(A,2I3)')' MC60A failed with INFO=',INFO
          STOP
        END IF

        READ(5,*) ALG
        IF (ALG.EQ.'S') THEN
C Work with supervariables
          CALL MC60B(N,LIRN,IRN,ICPTR,NSUP,SVAR,VARS,IW)
          WRITE(6,'(A,5I4)') 'The number of supervariables is', NSUP
          CALL MC60C(N,NSUP,LIRN,IRN,ICPTR,VARS,JCNTL,PERMSV,WEIGHT,
     *           PAIR,INFO,IW,W)
          CALL MC60F(N,NSUP,LIRN,IRN,ICPTR,VARS,PERMSV,IW,RINFO)
          CALL MC60D(N,NSUP,SVAR,VARS,PERMSV,PERM,IW)
        ELSE
C Work with variables
          NSUP = N
          DO 10 I = 1,N
            VARS(I) = 1
   10     CONTINUE
          CALL MC60C(N,NSUP,LIRN,IRN,ICPTR,VARS,JCNTL,PERM,WEIGHT,
     *            PAIR,INFO,IW,W)
          CALL MC60F(N,NSUP,LIRN,IRN,ICPTR,VARS,PERM,IW,RINFO)
        END IF

        WRITE(6,'(A,5I4)') 'The chosen permutation is', PERM
        WRITE(6,'(A,F4.0,/)') 'The profile is', RINFO(1)

   20 CONTINUE

       END
```

Suppose we wish to reduce the profile of a matrix with the following sparsity pattern:

$$\begin{pmatrix} x & x & x & x & x \\ x & x & x & & \\ x & x & x & & \\ x & & & x & \\ x & & & & x \end{pmatrix}$$

The following input data works with and then without supervariables. In each case, the lower-triangular part is provided column by column.

```
5 10
1  2  3  4  5  2  3  3  4  5
1  6  8  9 10 11
Variables

5 10
1  2  3  4  5  2  3  3  4  5
1  6  8  9 10 11
Supervariables
```

This produces the output:

```
The chosen permutation is   3   5   4   1   2
The profile is 10.

The number of supervariables is   4
The chosen permutation is   3   5   4   1   2
The profile is 10.
```

The pattern of the reordered matrix is:

$$\begin{pmatrix} x & & x & & \\ & x & x & & \\ x & x & x & x & x \\ & & x & x & x \\ & & x & x & x \end{pmatrix}$$