1 SUMMARY

Given a sparse symmetric matrix A, this routine combines a matching algorithm with a fill-reducing ordering algorithm to compute an **elimination order** that is suitable for use with a sparse direct solver. **Scaling factors** for A are optionally computed. The scaling factors s_i, $i = 1, \ldots, n$, are returned so that the scaled matrix is SAS where $S = \mathrm{diag}(s)$.

HSL_MC80 is recommended for computing an ordering and scaling for tough symmetric **indefinite** systems (which may be singular) since it aims to permute large off-diagonal entries on to the subdiagonal to provide good initial pivots when used with a sparse direct solver.

Note that the computed ordering may result in the analyse phase of the direct solver predicting more entries in the factors than for an ordering computed using nested dissection or minimum degree applied direct to A. However, if combined with the computed scaling of A, the HSL_MC80 ordering can often be used by the direct solver without modification to compute a numerically stable factorization, so that the actual number of entries in the factor and operations used to compute it may be less than for nested dissection or minimum degree applied to A.

ATTRIBUTES — Version: 1.1.4 (1 November 2023). **Interfaces:** Fortran, MATLAB. **Types:** Real (single, double), Complex (single, double). **Original date:** 20 June 2012. **Uses:** HSL_MC34, MC64, HSL_MC68 (optionally using METIS version 4.x). **Origin:** J.D. Hogg and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95. **Remark:** The development of HSL_MC80 was supported by the EPSRC grant EP/I013067/1.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement.

Single precision version
```
USE HSL_MC80_single
```
Double precision version
```
USE HSL_MC80_double
```
Complex version
```
USE HSL_MC80_complex
```
Double complex version
```
USE HSL_MC80_double_complex
```

2.2 Argument lists and calling sequences

2.2.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that optional arguments be called by keyword, not by position**.

2.2.2 Package type

We use the term **package type** to mean default real if the single precision version is being used, double precision real for the double precision version, default complex for the complex version, and double precision complex for the double complex version.

2.3 METIS

The HSL_MC80 package optionally uses the METIS graph partitioning library available from the University of Minnesota website. If METIS is not available, the user must link with the supplied dummy subroutine METIS_NodeND. In this case, the METIS ordering option will not be available to the user and, if selected, an error will be returned.

Important: At present, HSL_MC80 only supports METIS version 4, not the latest version 5 releases.

2.3.1 To compute an ordering and, optionally, scaling factors

If the user has the **lower triangular part of** A held by columns, a call of the following form should be made:

```
call mc80_order(ord,n,ptr,row,val,order,control,info[,scale,perm])
```

If the user has the **lower and upper triangular parts of** A held in compressed sparse column format, a call of the following form should be made:

```
call mc80_order_full(ord,n,ptr,row,val,order,control,info[,scale,perm])
```

ord is a scalar INTENT(IN) argument of type default INTEGER that controls the choice of ordering algorithm that is applied to the compressed matrix. Possible values are:

- 1 An approximate minimum degree ordering is used.
- 2 A minimum degree ordering is used.
- 3 METIS ordering is used.

n is a scalar INTENT(IN) argument of type default INTEGER that holds the order of A. **Restriction:** n ≥ 0.

ptr is an INTENT(IN) rank-one array of type default INTEGER and size n+1 that must be set by the user so that ptr(j) is the position in row of the first entry in column j (j=1,2,...,n) and ptr(n+1) must be set to one more than the total number of entries.

row is an INTENT(IN) rank-one array of type default INTEGER. On a call to mc80_order, it must be set so that row(1:ptr(n+1)-1) holds the row indices of the entries in the **lower triangular** part of A; on a call to mc80_order_full, it must be set so that row(1:ptr(n+1)-1) holds the row indices of the entries in the **lower and upper triangular** parts of A. The entries of a single 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 columns. Row indices within a column may be in any order. Zeros on the diagonal need not be included as entries with value zero.

val is an INTENT(IN) rank-one array of package type. It must be set so that val(k) holds the value of the entry in row(k), (k=1,2,...,ptr(n+1)-1).

order is an INTENT(OUT) rank-one array of type default INTEGER and size n. On exit, |order(i)| holds the position of variable i in the elimination order (pivot sequence). If a 1×1 pivot i is obtained, order(i)$>$0. If a 2×2 pivot involving i and j is obtained, order(i)$<$0, order(j)$<$0 and |order(j)|=|order(i)|+1.

control is a scalar INTENT(OUT) argument of type mc80_control. Its components control the execution of the subroutine, as explained in Section 2.3.2.

info is a scalar INTENT(OUT) argument of type mc80_info. Its components provide information about the execution of the subroutine, as explained in Section 2.3.3.

scale is an optional INTENT(OUT) rank-one array of package type and size n. On exit, scale(i) holds the scaling factor for row/column *i* of *A*. If the matching algorithm finds *A* to be structurally singular, the value of scale(j) for each unmatched row/column *j* of *A* depends on the value of control%zero_singular.

perm is an optional INTENT(OUT) rank-one array of type INTEGER and size n. On exit, perm holds the matching. Row i is matched to column perm(i); if i is unmatched, perm(i)=-1.

2.3.2 The derived data type for controlling the execution

The derived data type mc80_control is used to control the execution of mc80_order. The only component is:

action is a scalar of type LOGICAL. If set to .false. and the matrix is found to be structurally singular, the code exits immediately with an error flag set. Otherwise, a warning is issued and a scaling and ordering computed. The default is .true..

unmatched_scale_zero is a scalar of type LOGICAL and determines the scaling used for structually singular matrices. The rows and columns of a structually singular matrices are split into two sets: matched and unmatched. The matched set represents a structually non-singular submatrix of maximum rank on which the scaling is defined normally. If unmatched_scale_zero = .true. then the scaling for the unmatched rows and columns is set to zero. This will accelerate any subsequent factorization. However if the matched submatrix is numerically singular the resulting solution may be incorrect. Otherwise, if unmatched_scale_zero = .false. then the scaling is calculated such that entries in the unmatched part are less than or equal to one in absolute values. The default is .false..

unmatched_last is a scalar of type LOGICAL. If .true. and the matrix is structually singular, then all unmatched columns are ordered in positions info%struct_rank + 1,...n. Otherwise, if .false. unmatched columns are ordered in a position to minimise fill. The default is .false..

2.3.3 The derived data type for holding information

The derived data type mc80_info is used to hold information from the execution of mc80_order. The components are:

compress_rank is a scalar of type default INTEGER that holds the order of the compressed matrix (the ordering algorithm is applied to this matrix).

flag is a scalar of type default INTEGER that gives the exit status of the algorithm (details in Section 2.4).

flag68 is a scalar of type default INTEGER that holds the exit status of HSL_MC68.

max_cycle is a scalar of type default INTEGER that holds the length of the longest cycle (before splitting).

stat is a scalar of type default INTEGER that, in the event of an allocation error, holds the Fortran stat parameter if it is available (and is set to 0 otherwise).

struct_rank is a scalar of type default INTEGER that holds the structural rank of *A* that is computed by HSL_MC84.

2.4 Warning and error messages

A successful return is indicated by info%flag having the value zero. A negative value is associated with error as follows:

−1 Allocation error.

−2 n is negative.

−3 control%action=.false. and A found to be structurally singular.

−4 Unexpected error from HSL_MC68. The user is advised check the input matrix data (using, for example, HSL_MC69). Further information may be provided by info%flag68.

−5 ord is out of range.

−6 METIS ordering was requested but METIS is not available.

A positive value of info%flag is used to warn the user. Possible values are:

+1 control%action=.true. and A found to be structurally singular (see info%compress_rank).

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: HSL_MC34, MC64, HSL_MC68.

Restrictions: n≥0; ord=0,1,2,3.

4 METHOD

Explicit zeros within the supplied matrix are first removed, the absolute values of the matrix entries are taken and then, if the user has supplied only the lower triangular part of A, HSL_MC34 is used to expand the matrix. MC64 is used to compute a weighted matching and, if requested, the scaling factors are returned in scale (these scaling factors are the exponential of those returned by MC64).

The lengths of the cycles within the matching are next considered. Any cycles of length $2k$ ($k > 1$) are broken down into k cycles of length 2 and any cycles of length $2k+1$ ($k > 0$) are broken down into k cycles of length 2 and one of length 1. The pattern of the matrix is then condensed, with a single row/column replacing each pair of rows/columns in a cycle of length 2. The pattern of a row/column in the condensed matrix is the union of the rows/columns it replaces. HSL_MC68 is called to compute an ordering for the condensed matrix. Finally, this is translated back to an ordering for the original matrix (with a 2×2 pivot corresponding to each cycle of length 2).

4.1 Structually singular matrices

If A is structually singular, a non-singular submatrix $A_{I\times I}$ is identified by MC64. An optimal weighted matching is then obtained on this submatrix using a second call to MC64. Note that the set I is not chosen to be optimal in any way.

If $A_{I\times I}$ is numerically non-singular then it provides sufficient information to solve a consistent set of equations $Ax = b$. As such, rows and columns not in I may be ignored. The options control%unmatched_scale_zero and control%unmatched_last facilitate this. However, as careful handling is required if $A_{I\times I}$ is numerically singular, these options are disabled by default.

5 EXAMPLE OF USE

Consider the symmetric indefinite sparse matrix:

$$\begin{pmatrix} & 2.0\times10^{-6} & 1.5 & 1.1 & \\ 2.0\times10^{-6} & 0.2 & 1.2 & & \\ 1.5 & 1.2 & & 3.0 & \\ 1.1 & & 3.0 & & -1.0\times10^{-3} \\ & & & -1.0\times10^{-3} & \end{pmatrix}$$

Suppose we wish to find the elimination ordering and scaling generated by the matching algorithm with the approximate minimum degree method applied to the compressed matrix. The following code may be used:

```
program hsl_mc80ds
   use hsl_mc80_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   type (mc80_control) :: control
   type (mc80_info) :: info

   integer :: n, ne
   integer,  dimension (:), allocatable :: row, ptr, order
   real(wp), dimension (:), allocatable :: val, scale

   integer :: i, j
   real(wp), dimension(:, :), allocatable :: origA, scaledA

   ! Read in the order n of the matrix and number of entries in lower triangle.
   read (5,*) n, ne

   ! Allocate arrays
   allocate (row(ne),ptr(n+1),order(n),val(ne),scale(n))

   ! Read in matrix data
   read (5,*) ptr(1:n+1)   ! column starts
   read (5,*) row(1:ne)    ! row indices
   read (5,*) val(1:ne)    ! entries

   ! Compute elimination order using AMD on compressed matrix
   call mc80_order(1,n,ptr,row,val,order,control,info,scale=scale)
   if (info%flag < 0) then
      write (*,'(a,i3)') ' Error. info%flag = ', info%flag
      stop
   endif

   write (*,'(a,i3)') ' Exit mc80_order with info%flag = ', info%flag
   write (*,'(a)') ' Matching ordering combined with AMD: '
   write (*,'(8i4)') order
```

```
   write (*,'(a)')  ' Scaling factors:'
   write (*,'(8es12.4)') scale

   !
   ! Print original and scaled matrices
   !
   allocate(origA(n,n), scaledA(n,n))
   origA(:,:) = 0.0; scaledA(:,:) = 0.0
   do i = 1, n
      do j = ptr(i), ptr(i+1)-1
         origA(row(j), i) = val(j)
         origA(i, row(j)) = val(j)
         scaledA(row(j), i) = scale(i) * val(j) * scale(row(j))
         scaledA(i, row(j)) = scale(i) * val(j) * scale(row(j))
      end do
   end do

   write (*,'(/,a)') ' Original matrix:'
   do i = 1, n
      write(*, "(10f10.3)") origA(i,:)
   end do
   write (*,'(/,a)') ' Scaled matrix:'
   do i = 1, n
      write(*, "(10f10.3)") scaledA(i,:)
   end do
end program hsl_mc80ds
```

With the input data:

```
  5   7
  1   4   5   7   8   8
  2   3   4   2   3   4  5
  2.0e-6  1.5  1.1  0.2  1.2   3.0 -1.0e-3
```

this produces the output:

```
 Exit mc80_order with info%flag =   0
 Matching ordering combined with AMD:
  -4   3  -5  -1  -2
 Scaling factors:
  1.1547E+00  2.2361E+00  5.7735E-01  5.7735E-01  1.7321E+03

 Original matrix:
     0.000     0.000     1.500     1.100     0.000
     0.000     0.200     0.000     0.000     0.000
     1.500     0.000     1.200     3.000     0.000
     1.100     0.000     3.000     0.000    -0.001
     0.000     0.000     0.000    -0.001     0.000

 Scaled matrix:
```

```
 0.000     0.000     1.000     0.733     0.000
 0.000     1.000     0.000     0.000     0.000
 1.000     0.000     0.400     1.000     0.000
 0.733     0.000     1.000     0.000    -1.000
 0.000     0.000     0.000    -1.000     0.000
```