Science and Technology Facilities Council

HSL_MC73

PACKAGE SPECIFICATION HSL

1 SUMMARY

Let A be an $n \times n$ matrix with a symmetric sparsity pattern. HSL_MC73 has entries to compute the (approximate) **Fiedler vector** of the unweighted or weighted Laplacian matrix of A and to compute a symmetric permutation that reduces the **profile and wavefront of A by using a multilevel algorithm**. A number of profile reduction algorithms are offered:

(1) The multilevel algorithm of Hu and Scott [1] (referred to here as the multilevel Sloan algorithm),

(2) A multilevel **spectral ordering** algorithm, and

(3) A hybrid algorithm that refines the multilevel spectral ordering (2) using MC60.

In each case, an option exists to refine the computed ordering using the Hager exchange algorithm (MC67).
If Hager exchanges are not employed, the orderings computed using (1) and (3) generally yield smaller profiles and wavefronts than the spectral ordering (2). For some problems, (1) yields smaller profiles and wavefronts than (3), but for others the converse is true. Algorithm (1) is faster than (3). Using Hager exchanges can substantially increase the ordering cost but can give worthwhile reductions in the profile and wavefront.

[1] Y.K. Hu and J.A. Scott (2001). A multilevel algorithm for wavefront reduction, *SIAM J. Scientific Computing*, **23**, 1352-1375.

ATTRIBUTES — Version: 2.8.2 (1 November 2023). **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Uses:** HSL_MC65, HSL_ZD11, FA14, KB07, MC60, MC61, MC67, _AXPY, _NRM2, _COPY, _DOT. **Language:** Fortran 2003 subset (F95 + TR15581). **Original date:** October 2002. **Origin:** Y.F. Hu, Daresbury Laboratory and J.A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement of the form

Single precision version

```
USE HSL_MC73_SINGLE
```

Double precision version

```
USE HSL_MC73_DOUBLE
```

In HSL_MC73_SINGLE, all reals are default reals. In HSL_MC73_DOUBLE, all reals are double precision reals. In both modules, all integers are default integers.
The following procedures are available to the user:

(a) The initialization subroutine MC73_initialize may be called to set default values for the control parameters.

(b) If the user wishes to compute the (approximate) Fiedler vector, MC73_FIEDLER should be called.

(c) To compute a symmetric permutation that aims to reduce the profile, MC73_ORDER should be called.

(d) MC73_PRINT_MESSAGE can be used after a return from MC73_FIEDLER or MC73_ORDER to print the error or warning message associated with a nonzero error flag.

2.2 The control derived data type

The derived data type MC73_CONTROL is used to control the action. The user must declare a structure of type MC73_CONTROL. The components are automatically given default values in the definition of the type. The components may also be initialized to their default values by a call to MC73_INITIALIZE; the user does not need to set them unless values other than the defaults are required. The following components are employed:

LP is an INTEGER scalar used as the output stream for error messages. If it is negative, these messages will be suppressed. The default value is 6.

WP is an INTEGER scalar used as the output stream for warning messages. If it is negative, these messages will be suppressed. The default value is 6.

MP is an INTEGER scalar used as the output stream for diagnostic messages. If it is negative, these messages will be suppressed. The default value is 6.

PRINT_LEVEL is an INTEGER scalar indicating the level of diagnostic printing desired.

- <0 no printing.
- 0 error and warning messages only.
- 1 as 0 plus basic diagnostic messages.
- 2 as 1 plus some more detailed diagnostic messages.

The default value is 0. Values greater than 2 are treated as 2.

The remaining control parameters are only likely to be of interest to the more experienced user.

MGLEVEL is an INTEGER scalar that holds the maximum number of levels in the multilevel hierarchy. If MGLEVEL = 1 and JOB = 1 on a call to MC73_ORDER, the ordering is computed using MC61. Values less than 1 are treated as 1. The default value is 100.

COARSEST_SIZE is an INTEGER scalar that holds the problem size in the multilevel hierarchy after which no further coarsening is performed. The default value is 200. Values less than 2 are treated as 2.

MAX_REDUCTION and MIN_REDUCTION are REAL scalars that hold the maximum and minimum grid reduction factors. If two successive grids have n_c and n_f nodes, respectively, coarsening continues while $n_c < n_f$*MAX_REDUCTION and $n_c > n_f$*MIN_REDUCTION. MAX_REDUCTION must be at least 0.5 and at most one 1; values less than 0.5 are treated as 0.5 and those greater than 1 are treated as 1. MIN_REDUCTION must be greater than zero and less than MAX_REDUCTION; values outside this range are replaced by the default. The default values are 0.8 and 0.1, respectively. These parameters are not used if MGLEVEL = 1.

MLANCZ is an INTEGER scalar that holds the maximum number of Lanczos vectors used by the Lanczos algorithm on the coarsest grid. The default value is 300.

TOL is a REAL scalar that holds the convergence tolerance for the Lanczos algorithm on the coarsest grid. The Lanczos algorithm is used to compute eigenpairs and TOL is a tolerance on the residual of the computed eigenpair. The default value is 10^{-3}. Small values may slow convergence while providing greater accuracy. TOL is not used on a call to MC73_ORDER with JOB = 1.

TOL1 is a REAL scalar that holds the convergence tolerance for the Rayleigh Quotient iterations at each refinement step. The default value is 10^{-3}. TOL1 is not used on a call to MC73_ORDER with JOB = 1.

RTOL is a REAL scalar that determines the convergence tolerance for the SYMMLQ algorithm used for solving linear systems within the Rayleigh Quotient iterations. The default value is 10^{-2}. The convergence tolerance used by SYMMLQ within MC73_FIEDLER is min(10^{-3}*RTOL, 10*TOL) and within MC73_ORDER it is min(RTOL, 10*TOL). A smaller value generally gives a more accurate Fiedler vector but may take longer to compute. RTOL is not used on a call to with JOB = 1.

MAXIT is an INTEGER scalar that holds the maximum number of Rayleigh Quotient iterations at each refinement step. The default value is 10. MAXIT is not used on a call to MC73_ORDER with JOB = 1.

HAGER_EXCHANGE controls the use of the Hager exchange algorithms. If HAGER_EXCHANGE $\leq$ 0, exchange algorithms are not used. Otherwise, down/up exchanges are applied a maximum of HAGER_EXCHANGE times. The default value is 0. HAGER_EXCHANGE is not used on a call to MC73_FIEDLER or on a call to MC73_ORDER with WGT present.

3 THE ARGUMENT LISTS

We use square brackets [ ] to indicate OPTIONAL arguments.

3.1 To initialize the control variables

```
CALL MC73_INITIALIZE(CONTROL)
```

CONTROL is a scalar of type MC73_CONTROL of INTENT(OUT). On exit, its components will have been given the default values specified in Section 2.2.

3.2 To compute an approximate Fiedler vector

```
CALL MC73_FIEDLER(N,LIRN,IRN,IP,LIST,FVECTOR,CONTROL,INFO[,WGT])
```

N is an INTEGER scalar with INTENT(IN). On entry it must hold the order n of A. **Restriction:** N $\geq$ 1.

LIRN is an INTEGER scalar with INTENT(IN). LIRN must be at least the number of entries input by the user in IRN. **Restriction:** LIRN $\geq$ IP(N+1)-1.

IRN is an INTEGER array of rank one with INTENT(IN) and size LIRN. It must be set by the user to hold the row indices of the entries in the strictly lower triangular part of A, with the entries in column 1 preceding those in column 2, and so on, with no space between columns. Within each column, the entries may be in arbitrary order. If IRN(k) is less than 1 or greater than N, the entry is ignored. Entries in the upper triangular part of A are allowed. In this case, the corresponding lower triangular entries are treated as nonzero, even if not explicitly entered by the user. Duplicated entries are permitted. Diagonal entries are ignored.

IP is an INTEGER array of rank one with INTENT(IN) and size N+1. It must be set by the user to hold the starting positions of the columns of the lower triangular part of A in compressed sparse column storage format. More specifically, the row indices of column J must be in IRN(IP(J):IP(J+1)-1).

LIST is INTEGER array of rank one with INTENT(OUT) and size N. On exit, the number of the component within the adjacency graph of A that variable I belongs to is LIST(I), I = 1, 2, ..., N. If a component consists of a single vertex, the corresponding entry in LIST is assigned a value 0.

FVECTOR is a REAL array with INTENT(OUT) and size N. On exit, FVECTOR holds the Fiedler vector of A. If the adjacency graph of A has more than one component (see INFO(2)), the Fiedler vector for each component is computed and the resulting vectors agglomerated in FVECTOR. If a component consists of a single vertex, the corresponding variable is assigned a value 0 in FVECTOR.

CONTROL is a scalar of type MC73_CONTROL with INTENT(IN). Its components control the action, as explained in Section 2.2.

INFO is an INTEGER array of rank one with INTENT(OUT) and size 10. INFO(1) is used as an error/warning flag. Negative values indicate an error and positive values a warning. For nonzero values of INFO(1), see section 3.6. For details of the information contained in the other components of INFO, see section 3.5.

WGT is an OPTIONAL REAL array with INTENT(IN) and size at least IP(N+1)-1. If present, the Fiedler vector of the weighted Laplacian of A is computed. WGT must be set by the user so that |WGT(k)| holds the value of the weight corresponding to the entry of A whose row index is held in IRN(k). Duplicated entries are summed. Diagonal entries are ignored.

3.3 To compute a symmetric permutation

```
CALL MC73_ORDER(JOB,N,LIRN,IRN,IP,PERM,CONTROL,INFO,RINFO[,WGT])
```

JOB is an INTEGER scalar of INTENT(IN). On entry, it must be set by the user to indicate which algorithm is required. The options are:

- 1: multilevel Sloan algorithm
- 2: multilevel spectral ordering algorithm
- 3: hybrid ordering algorithm (spectral ordering refined using MC60)

Restriction: JOB = 1, 2, 3.

N, LIRN, IRN, IP are all as in the call to MC73_FIEDLER.

PERM is INTEGER array of rank one with INTENT(OUT) and size N. On exit, the new ordering is contained in PERM. The position of variable I in the new ordering is PERM(I), I = 1, 2, ..., N.

CONTROL is a scalar of type MC73_CONTROL with INTENT(IN). Its components control the action, as explained in Section 2.2.

INFO is an INTEGER array of rank one with INTENT(OUT) and size 10. INFO(1) is used as an error/warning flag. Negative values indicate an error and positive values a warning. For nonzero values of INFO(1), see section 3.6. For details of the information contained in the other components of INFO, see section 3.5.

RINFO is a REAL array of rank one with INTENT(OUT) and size 20. For details of the information contained in RINFO on successful exit, see section 3.5.

WGT is an OPTIONAL REAL array with INTENT(IN) and size at least IP(N+1)-1. WGT is not accessed if JOB = 1. Otherwise, if present, in the computation the spectral ordering the Fiedler vector of the weighted Laplacian of A is computed. It must be set by the user so that |WGT(k)| holds the value of the weight corresponding to the entry of A whose row index is held in IRN(k). Duplicated entries are summed. Diagonal entries are ignored.

3.4 Printing error/warning messages

Subroutine MC73_PRINT_MESSAGE can be used after return from MC73_FIELDER or MC73_ORDER to print an error or warning message associated with a nonzero error flag INFO.

```
CALL MC73_PRINT_MESSAGE(FLAG,[,UNIT,MESSAGE])
```

FLAG is an INTEGER scalar of INTENT(IN). On entry, it must be set by the user to hold the error flag generated when calling MC73_FIELDER or MC73_ORDER.

UNIT is an OPTIONAL INTEGER scalar of INTENT(IN). If present, on entry, it must be set by the user to hold the unit number for the error or warning message. If this number is negative, printing is suppressed. If UNIT is not present, the message will be printed on unit 6.

MESSAGE is an OPTIONAL CHARACTER (LEN=*) scalar with INTENT(IN). If present, on entry, it must be set by the user to hold the message to be printed ahead of the error or warning message.

3.5 Information arrays

The elements of the arrays INFO and RINFO provide information.

INFO(1) is used as an error/warning flag. Negative values indicate an error and positive values a warning. For nonzero values of INFO(1), see section 3.6. If an error is detected, the information contained in INFO(2:5) and in RINFO may be incomplete.

INFO(2) holds the number of non-trivial components (components with more than one vertex) in the adjacency graph of A.

The remaining entries of INFO are currently not used.

RINFO(1:4) hold the profile, maximum wavefront, semibandwidth, and root mean squared wavefront of A.

RINFO(5:8) hold the profile, maximum wavefront, semibandwidth, and root mean squared wavefront for the ordering held in PERM.

RINFO(9:12) hold the profile, maximum wavefront, semibandwidth, and root mean squared wavefront for the spectral ordering (JOB = 3 only).

The remaining entries of RINFO are currently not used.

3.6 Error diagnostics

On successful completion, the subroutines in the HSL_MC73 module will exit with the parameter INFO(1) set to 0. Other values for INFO(1) and the reasons for them are given below.

A negative value for INFO(1) is associated with a fatal error. Possible negative values for INFO(1) are:

-1 memory allocation failed.

-2 memory deallocation failed.

-3 N ≤ 0.

-4 LIRN is too small.

-5 IP is not monotonically increasing.

-6 JOB has an invalid value (MC73_ORDER only).

-7 Size of array A is too small.

A positive value for INFO(1) is associated with a warning. Possible positive values for INFO(1) are:

+1 one or more entries in IRN was found to be out of range.

+2 duplicate entries were found.

+4 the maximum number of Rayleigh Quotient iterations was reached.

+8 the Hager exchange algorithm did not reduce the profile.

Positive values of INFO(1) are summed so that, on exit, the user can identify all warnings, e.g. INFO(1) = 3 indicates both warnings 1 and 2 are raised.

4 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: FA14I/ID, FA14A/AD, KB07A/AD, MC60C/CD, MC60F/FD, MC61I/ID, MC61A/AD, MC67I/ID, MC67A/AD. BLAS routines SAXPY/DAXPY, SNRM2/DNRM2, SCOPY/DCOPY, SDOT/DDOT.

Other modules used directly: HSL_MC65, HSL_ZD11.

Input/output: Error, warning and diagnostic messages only. Error messages on unit CONTROL%LP and warning and diagnostic messages on units CONTROL%WP and CONTROL%MP, respectively. These have default value 6; printing of these messages is suppressed if the relevant unit number is negative or if PRINT_LEVEL is negative.

Changes from Version 1.0.0: HSL_ZD11 used instead of HSL_ZD01.

Restrictions: N $\geq$ 1, LIRN $\geq$ IP(N+1)-1, JOB = 1, 2, 3.

Portability: Fortran 2003 subset(F95+TR15581).

5 METHOD

Both MC73_FIEDLER and MC73_ORDER first use HSL_MC65 to check the user's data and construct the sparsity pattern of the whole of A. If MC73_ORDER is called, the algorithm implemented depends upon the choice of the parameter JOB. For each value of JOB, the ordering is optionally refined by the HSL routine MC67A/AD, which implements Hager's exchange algorithm [3]. Note that only down/up exchanges are offered; if any of the alternative options offered by MC67 are wanted, the user should call MC67A/AD with the appropriate choice of control parameters after the return from MC73_ORDER.

JOB = 1

The adjacency graph $G(A)$ of A is first constructed. We assume initially that the adjacency graph $G(A)$ of A has a single component. A series of graphs of successively smaller sizes is generated, using a maximal independent vertex set for the coarsening. An independent set of vertices is a subset of the vertices such that no two vertices in the subset are connected by an edge in the graph. An independent set is maximal if the addition of an extra vertex always destroys the independence. The smallest graph is reordered using Sloan's algorithm (MC60C/CD is used for this). To map the ordering on a coarse grid onto the next level, the vertices of the fine graph are given global priority values by mapping the coarse graph ordering onto the fine graph. This is then refined by the second phase of Sloan's algorithm, again using MC60C/CD. Two sets of weights (2,1) and (16,1) are used and the ordering with the smallest profile is selected. Full details are given by Hu and Scott [1]. Note that if the user sets the number of levels to 1 (that is, CONTROL%MGLEVEL = 1), MC61 is called to implement Sloan's algorithm.

If the adjacency graph $G(A)$ of A has more than one component, an ordering for each component is computed in turn. The computed ordering starts with the components that consist of single vertices and is followed in turn by the orderings of each of the non-trivial components.

`MC73_FIEDLER` and `JOB = 2`

The Laplacian matrix L corresponding to the matrix A is first computed. If the optional argument `WGT` is **not** present, the unweighted Laplacian of A is used. In this case, L has entries l_{ij} given by

$$\{l_{ij}\} = \begin{cases} -1 & \text{if } i \neq j \text{ and } \quad a_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad a_{ij} = 0 \\ \sum_{j:i\neq j} |l_{ij}| & \text{if } i = j. \end{cases} \tag{5.1}$$

If the optional argument `WGT` is present, then the weighted Laplacian of A is used. This has entries

$$\{l_{ij}\} = \begin{cases} -|w_{ij}| & \text{if } i \neq j \text{ and } \quad a_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad a_{ij} = 0 \\ \sum_{j:i\neq j} l_{ij} & \text{if } i = j. \end{cases} \tag{5.2}$$

A series of problems of smaller and smaller sizes is then generated, using heavy-edge matching for the coarsening. Edge collapsing selects pairs of adjacent vertices and each pair is coalesced into one new vertex. Each edge of the fine graph has weight one. Each edge of the coarse graph also has a weight associated with it which is related to the number of edges in the fine graph that it replaces; heavy-edge matching preferentially collapses heavier edges. The Lanczos algorithm is used on the smallest problem to compute its Fiedler vector, that is, the eigenvector corresponding to the smallest positive eigenvalue. This vector is then projected from one level to another as follows:

- The approximate Fiedler vector for the previous level is interpolated to provide an approximation to the Fiedler vector for the current level.
- The approximate Fiedler vector is then refined using the Rayleigh Quotient Iteration algorithm to give a more accurate result. The SYMMLQ algorithm is used for solving linear systems within the Rayleigh Quotient Iterations.

In `MC73_ORDER`, the entries of the final refined Fiedler vector are sorted in nondecreasing order to give the spectral ordering. If the adjacency graph of A has more than one component, a spectral ordering for each component is computed in turn. Any components consisting of a single vertex are ordered first.

In `MC73_FIEDLER`, the approximate Fiedler vector is returned without sorting. If the adjacency graph has more than one component, the approximate Fiedler vector for each component is computed and the resulting Fiedler vectors agglomerated in the array `FVECTOR`. The array `LIST` is used to indicate the number of the component to which each variable belongs. If a component consists of a single vertex, it is assigned a value 0 in `LIST` and in the returned Fiedler vector. Further details may be found in Barnard and Simon [2].

`JOB = 3`

The algorithm proceeds as in the case `JOB = 2`. Once the spectral ordering has been computed, `MC60C/CD` is called to compute a hybrid ordering. Two sets of weights (1,2) and (16,1) are used and the ordering with the smallest profile is selected. If different weights are wanted, `MC60C/CD` may be called with the user's choice of weights after the spectral ordering has been computed.

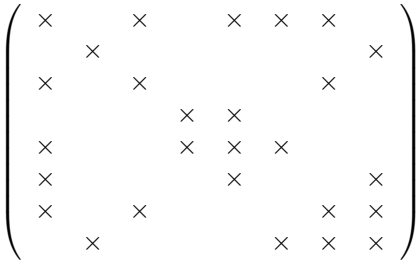
[1] Y.K. Hu and J.A. Scott (2001). A multilevel algorithm for wavefront reduction, *SIAM J. Scientific Computing*, **23**, 1352-1375.

[2] S.T. Barnard and H.D. Simon (1994). A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, **6**, 101-117.

[3] W.W Hager (2002). Minimizing the profile of a symmetric matrix. *SIAM Journal on Scientific Computing*, **23**, 1799-1816.

6 EXAMPLE OF USE

We illustrate the use of MC73 on the following matrix:



Program

```
   program example
      use hsl_mc73_double

      integer, parameter :: wp = kind(0.0d0)

! mc73 controller:
      type (mc73_control) :: control
! n: number of rows.
! nz: number of input entries.
! lirn: length lirn
! irn: row indices of the matrix
! ip: column pointers
      integer :: n,nz
      integer :: lirn
      integer, allocatable :: irn(:)
      integer, allocatable :: ip(:)
      integer, allocatable :: perm(:)
      real (kind = wp) :: rinfo(20)
      integer :: info(1:10)

      integer job
      integer st

      read (5,*) n,lirn
      allocate(irn(lirn),ip(n+1),perm(n),stat=st)
      if (st /= 0) then
         write (6,*) ' Allocation error'
         stop
      end if
      read (5,*) ip(1:n+1)
      nz = ip(n+1) - 1
      read (5,*) irn(1:nz)

! Reset control parameter
      control%coarsest_size = 2

! Compute hybrid ordering
      job = 3
      call mc73_order(job,n,lirn,irn,ip,perm,control,info,rinfo)
      if (info(1) < 0) then
        call mc73_print_message(info(1),6,"mc73_order")
        stop
      end if
```

```
      write (6,'(/a,8i4)') ' Profile before/after mc73   : ',   &
           int(rinfo(1)),int(rinfo(5))
      write (6,'(a,8i4)') ' Bandwidth before/after mc73 : ',   &
           int(rinfo(3)),int(rinfo(7))
      write (6,'(/a,8i4)') ' Permutation : ',perm
      deallocate(irn,ip,perm)

   end program example
```

The input data is

```
8 30
1  6  8  10  12  14  15  17  18
3  1  5  6  7  2  8  3  7  4  5  6  5  8  7  8  8
```

This produces the following output

```
 Profile before/after mc73   :   31  20
 Bandwidth before/after mc73 :    7   5

 Permutation :    5   3   1   8   7   6   2   4
```

The permuted matrix is

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