Science and Technology Facilities Council

# HSL_MC79

PACKAGE SPECIFICATION HSL

## 1 SUMMARY

Given the sparsity pattern of a rectangular sparse matrix $A = \{a_{ij}\}_{m\times n}$, HSL_MC79 has entries to compute a **maximum matching**, and a row permutation $P$ and column permutation $Q$ such that $PAQ$ is of **block triangular form**: a **coarse Dulmage-Mendelsohn decomposition** and a **fine Dulmage-Mendelsohn decomposition** is available.

A *matching* is a set of the rows $\mathcal{R}$ and columns $\mathcal{C}$, where each row in $i \in \mathcal{R}$ is paired with a unique $j \in \mathcal{C}$ subject to $a_{ij} \neq 0$. The size of a matching is defined to be equal to the number of columns in $\mathcal{C}$. A *maximum matching* of $A$ is a matching of $A$ that has size greater than or equal to any other matching of $A$. The size of the maximum matching is equal to the *structural rank* of the matrix.

The *Dulmage-Mendelsohn decomposition* consists of a row permutation $P$ and a column permutation $Q$ such that

$$PAQ = \begin{array}{c} \\ \mathcal{R}_1 \\ \mathcal{R}_2 \\ \mathcal{R}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \mathcal{C}_1 & \mathcal{C}_2 & \mathcal{C}_3 \end{array} \\ \begin{bmatrix} A_1 & A_4 & A_6 \\ 0 & A_2 & A_5 \\ 0 & 0 & A_3 \end{bmatrix} \end{array}, \tag{1.1}$$

where $A_1$, formed by the rows in the set $\mathcal{R}_1$ and the columns in the set $\mathcal{C}_1$, is an underdetermined matrix with $m_1$ rows and $n_1$ columns ($m_1 < n_1$ or $m_1 = n_1 = 0$); $A_2$, formed by the rows in the set $\mathcal{R}_2$ and the columns in the set $\mathcal{C}_2$, is a square, well-determined matrix with $m_2$ rows; $A_3$, formed by the rows in the set $\mathcal{R}_3$ and the columns in the set $\mathcal{C}_3$, is an overdetermined matrix with $m_3$ rows and $n_3$ columns ($m_3 > n_3$ or $m_3 = n_3 = 0$). In particular, let the set of rows $\mathcal{R}$ and the set of columns $\mathcal{C}$ form a maximum matching of $A$. The sets $\mathcal{R}_1$ and $\mathcal{R}_2$ are subsets of $\mathcal{R}$, and $\mathcal{R}_3 \cap \mathcal{R}$ has $n_3$ entries. The sets $\mathcal{C}_2$ and $\mathcal{C}_3$ are subsets of $\mathcal{C}$, and $\mathcal{C}_1 \cap \mathcal{C}$ has $m_1$ entries.

The *coarse Dulmage-Mendelsohn decomposition* orders the unmatched columns as the first columns in $PAQ$ and orders the unmatched rows as the last rows in $PAQ$. The output from the coarse Dulmage-Mendelsohn decomposition can be used to find a node separator from an edge separator of a graph [1].

The *fine Dulmage-Mendelsohn decomposition* computes a row permutation $P$ and a column permutation $Q$ such that $A_1$ and $A_3$ are block diagonal and each diagonal block is irreducible, and $A_2$ is block upper triangular with strongly connected (square) diagonal blocks. If $A$ is reducible and nonsingular, the fine Dulmage-Mendelsohn decomposition of a matrix $A$ can be used to solve the linear systems $Ax = b$ with block back-substitution.

[1] A. Pothen and C.-J. Fan (1990). *Computing the Block Triangular Form of a Sparse Matrix*, ACM Transactions on Mathematical Software, **16**, 303-324.

**ATTRIBUTES — Version:** 1.1.1 (1 November 2012). **Interfaces:** C, Fortran. **Types:** Integer. **Original date:** January 2011. **Origin:** H. S. Thorne, Rutherford Appleton Laboratory. **Language:** Fortran 2003 subset (F95 + TR 15581 + C interoperability). **Remark:** The development of this package was supported by EPSRC grant EP/E053351/1.

## 2 HOW TO USE THE PACKAGE

### 2.1 Calling sequences

Access to the package requires a USE statement of the form

```
USE HSL_MC79_integer
```

All integers are default integers.

The following subroutines are available to the user:

(a) To compute a maximum matching, MC79_matching should be called.

(b) To compute a coarse Dulmage-Mendelsohn decomposition, MC79_coarse should be called.

(c) To compute a fine Dulmage-Mendelsohn decomposition, MC79_fine should be called.

Each call accepts a sparse matrix that is stored using standard HSL format: this may be setup using the HSL_MC69 package, see Section 2.3.1.

### 2.2 The derived data types

The user must employ the derived types defined by the module to declare scalars of type MC79_control and MC79_info. The following pseudocode illustrates this.

```
use hsl_mc79_integer
...
type (mc79_control) :: control
type (mc79_info)    :: info
```

The components of MC79_control and MC79_info are described in Section 2.4.1 and Section 2.4.2, respectively.

### 2.3 Argument lists and calling sequences

#### 2.3.1 Input of the matrix $A$

The user must supply the matrix $A$ in standard HSL format. This is a compressed sparse column format with the entries within each column ordered by increasing row index. **No checks** are made on the user's data. It is important to note that any out-of-range entries or duplicates may cause HSL_MC79 to fail in an unpredictable way. Before using HSL_MC79, the HSL package HSL_MC69 may be used to check for errors and to handle duplicates (HSL_MC69 sums them) and out-of-range entries (HSL_MC69 removes them).

If the user's data is held using another standard sparse matrix format (such as coordinate format or sparse compressed row format), we recommend using a conversion routine from HSL_MC69 to put the data into standard HSL format. The input of $A$ is illustrated in Section 5.

#### 2.3.2 To compute the maximum matching

The method constructs a maximum matching for the matrix $A$.

```
CALL MC79_matching(m,n,ptr,row,rowmatch,colmatch,control,info)
```

m is an INTENT(IN) scalar of type INTEGER that must hold the number of rows in $A$. **Restriction:** m$\geq 1$.

n is an INTENT(IN) scalar of type INTEGER that must hold the number of columns in $A$. **Restriction:** n$\geq 1$.

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

row is an INTENT(IN) rank-one array of type INTEGER. The first ptr(n+1)-1 entries must hold the row indices of the entries of $A$, with the row indices for the entries in column 1 preceding those for column 2, and so on.

rowmatch is an INTENT(OUT) rank-one array of type INTEGER and size m. On exit, if rowmatch(i)=0, then row i of $A$ is not matched to any column. If rowmatch(i)=j and j>0, then row i of $A$ is matched to column j of $A$.

colmatch is an INTENT(OUT) rank-one array of type INTEGER and size n. On exit, if colmatch(j)=0, then column j of $A$ is not matched to any column. If colmatch(j)=i and i>0, then column j of $A$ is matched to row i of $A$.

control is a scalar of type MC79_control with INTENT(IN). Its components control the action, as explained in Section 2.4.1.

info is an INTENT(OUT) scalar of type MC79_info. Its components provide information about the execution of the subroutine, as explained in Section 2.4.2. In particular, info.flag is used as an error/warning flag, as detailed in Section 2.5.

### 2.3.3 To compute a coarse Dulmage-Mendelsohn decomposition

The method constructs the row and column permutation of a coarse Dulmage-Mendelsohn decomposition of $A$.

```
CALL MC79_coarse(m,n,ptr,row,rowperm,colperm,control,info)
```

m, n, ptr, row, control and info are all as in the call to MC79_matching.

rowperm is an INTENT(OUT) rank-one array of type INTEGER and size m. On exit, if rowperm(i)=j, then row j of $A$ becomes row i of the permuted matrix.

colperm is an INTENT(OUT) rank-one array of type INTEGER and size n. On exit, if colperm(i)=j, then column j of $A$ becomes column i of the permuted matrix.

### 2.3.4 To compute a fine Dulmage-Mendelsohn decomposition

The method constructs the row and column permutation of a fine Dulmage-Mendelsohn decomposition of $A$.

```
CALL MC79_fine(m,n,ptr,row,rowperm,colperm,rowptr,colptr,control,info)
```

m, n, ptr, row, control and info are all as in the call to MC79_matching.

rowperm and colperm are as in the call to MC79_coarse.

rowptr is an INTENT(OUT) rank-one array of type INTEGER and size m+2. On exit, it holds the index in the reordered matrix of the first row in each component:

rowptr(1),..., rowptr(info%hz_comps) give the indices of the first row (with respect to the permuted matrix) of each irreducible diagonal block that lies within $A_1$ of (1.1);

rowptr(info%hz_comps+1),...,rowptr(info%hz_comps+info%sq_comps) give the indices of the first row (with respect to the permuted matrix) of each strongly connected diagonal block that lies within $A_2$ of (1.1);

rowptr(info%hz_comps+info%sq_comps+1),...,rowptr(info%hz_comps+info%sq_comps+info%vt_comps) give the indices of the first row (with respect to the permuted matrix) of each irreducible diagonal block block that lies within $A_3$ of (1.1).

In addition, rowptr(info%hz_comps+info%sq_comps+info%vt_comps+1) is equal to m+1. The remaining entries are set to 0.

colptr is an INTENT(OUT) rank-one array of type INTEGER and size n+2. On exit, it holds the index in the reordered matrix of the first column in each component:

colptr(1),...,colptr(info%hz_comps) give the indices of the first column (with respect to the permuted matrix) of each irreducible diagonal block that lies within $A_1$ of (1.1);

colptr(info%hz_comps+1),...,colptr(info%hz_comps+info%sq_comps) give the indices of the first column (with respect to the permuted matrix) of each strongly connected diagonal block that lies within $A_2$ of (1.1);

colptr(info%hz_comps+info%sq_comps+1),...,colptr(info%hz_comps+info%sq_comps+info%vt_comps) give the indices of the first column (with respect to the permuted matrix) of each irreducible diagonal block block that lies within $A_3$ of (1.1).

In additon, colptr(info%hz_comps+info%sq_comps+info%vt_comps+1) is equal to n+1. The remaining entries are set to 0.

### 2.4 The derived data types

#### 2.4.1 The derived data type for holding control parameters

The derived data type MC79_control is used to control printing. The user must declare a structure of type MC79_control. Components of this derived type are automatically given their default values in the definition of the type: 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 that is used as the output stream for error messages. If it is negative, these messages will be suppressed. The default value is 6.

mp is an INTEGER scalar that is that is 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. The levels are:

<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.

#### 2.4.2 The derived data type for holding information

The derived data type MC79_info is used to hold information from the execution of MC79_matching, MC79_coarse and MC79_fine. The components are:

**Information returned by all subroutines**

flag is an INTEGER scalar used as an error flag (details in Section 2.5).

mbar is a scalar of type INTEGER that holds the number of rows that cannot be matched to columns in a maximum matching of $A$.

nbar is a scalar of type INTEGER that holds the number of columns that cannot be matched to rows in a maximum matching of $A$.

stat is a scalar of type INTEGER that holds the Fortran stat parameter.

**Information returned by `MC79_coarse` and `MC79_fine` only**

`m1` is a scalar of type `INTEGER` that holds the number of rows in the submatrix $A_1$, where $A_1$ is defined by equation (1.1).

`m2` is a scalar of type `INTEGER` that holds the number of rows in the submatrix $A_2$, where $A_2$ is defined by equation (1.1).

`m3` is a scalar of type `INTEGER` that holds the number of rows in the submatrix $A_3$, where $A_3$ is defined by equation (1.1).

`n1` is a scalar of type `INTEGER` that holds the number of columns in the submatrix $A_1$, where $A_1$ is defined by equation (1.1).

`n2` is a scalar of type `INTEGER` that holds the number of columns in the submatrix $A_2$, where $A_2$ is defined by equation (1.1).

`n3` is a scalar of type `INTEGER` that holds the number of columns in the submatrix $A_3$, where $A_3$ is defined by equation (1.1).

**Information returned by `MC79_fine` only**

`hz_comps` is a scalar of type `INTEGER` that holds the number of components found in $A_1$, where $A_1$ is defined by equation (1.1).

`sq_comps` is a scalar of type `INTEGER` that holds the number of components found in $A_2$, where $A_2$ is defined by equation (1.1).

`vt_comps` is a scalar of type `INTEGER` that holds the number of components found in $A_3$, where $A_3$ is defined by equation (1.1).

### 2.5 Warning and error messages

A successful return from a subroutine in the package is indicated by `info%flag` having the value zero. A negative value is associated with an error message that by default will be output on unit `control%lp`.
Possible negative values are:

`-1` Memory allocation failed. If available, the `stat` parameter is returned in `info.stat`.

`-2` Memory deallocation failed. If available, the `stat` parameter is returned in `info.stat`.

`-3` `n`$<$`0`.

`-4` `m`$<$`0`.

## 3 GENERAL INFORMATION

**Input/output:** Error, warning and diagnostic messages. Error messages on unit `control%lp` and diagnostic messages on unit `control%mp`. These have default value `6`; printing of these messages is suppressed if the relevant unit number is negative or if `print_level` is negative.

**Restrictions:** `m`$\geq$ `1` and `n`$\geq$ `1`.

**Portability:** Fortran 95 + TR15581.

## 4 METHOD

Let $G = (V,E)$ be the bipartite graph of $A = \{a_{ij}\}_{m\times n}$, with $m$ row vertices, $n$ column vertices, and undirected edges $E = \{(i,j)|a_{ij} \neq 0\}$. An *alternating augmenting path* is a path that starts at an unmatched column/row and traverses through the graph $G$ until it reaches an unmatched row/column, subject to every other edge in the path being matched.

MC79_matching constructs a maximum matching by performing depth-first searches from unmatched columns to find any alternating augmenting paths. When an alternating augmenting path is found, the unmatched edges in the path become matched edges and the previously matched edges become unmatched edges. The method continues until no more alternating augmenting paths can be found. This method is sometimes called the Ford-Fulkerson method [1].

The *Dulmage-Mendelsohn decomposition* consists of a row permutation $P$ and a column permutation $Q$ such that

$$PAQ = \begin{bmatrix} A_1 & A_4 & A_6 \\ 0 & A_2 & A_5 \\ 0 & 0 & A_3 \end{bmatrix},$$

where $A_1$ is a matrix with $m_1$ rows and $n_1$ columns $(m_1 \leq n_1)$, $A_2$ is a matrix with $m_2$ rows and $n_2$ columns $(m_2 = n_2)$, and $A_3$ is a matrix with $m_3$ rows and $n_3$ columns $(m_3 \geq n_3)$. The rows in $A_1$ correspond to rows from $A$ that lie in the set $I$, where

$$I = \{i : i \text{ is reachable from some } j \text{ via an alternating augmenting path, where } j \notin C\}.$$

The columns of $A_1$ correspond to the union of the set of unmatched columns from $A$ and the set of columns that are matched to rows in $I$. The columns in $A_3$ correspond to columns from $A$ that lie in the set $\mathcal{J}$, where

$$\mathcal{J} = \{j : j \text{ is reachable from some } i \text{ via an alternating augmenting path, where } i \notin \mathcal{R}\}.$$

The rows of $A_3$ correspond to the union of the set of unmatched rows from $A$ and the set of rows that are matched to columns in $\mathcal{J}$.

MC79_coarse starts by finding a maximum matching using the same methodology as MC79_matching. The method then proceeds to find the rows in $A_1$ by performing depth-first searches from the unmatched columns to find all of the row vertices that are reachable from the unmatched columns via alternating augmenting paths. The columns in $A_1$ are defined to be the union of the set of unmatched columns and the set of columns matched with the rows in $A_1$ (the unmatched columns are ordered first). Similarly, the columns in $A_3$ are found by performing depth-first searches from the unmatched rows to find all of the column vertices that are reachable from the unmatched rows via alternating augmenting paths. The rows in $A_3$ are defined to be the union of the set of unmatched rows and the set of rows matched to the columns in $A_3$ (the unmatched rows are ordered last).

MC79_fine proceeds as MC79_coarse until all of the rows and columns in $A_1$, $A_2$ and $A_3$ have been computed. The method then searches $A_1$ and $A_3$ to find any irreducible blocks and computes the permutation required to place these irreducible blocks on the diagonals of $A_1$ and $A_3$, respectively. Finally, the subroutine uses Tarjan's algorithm [2] to find the strongly connected components in $A_2$ and a permutation is formed to reduce $A_2$ to block upper triangular form (with the strongly connected components lying on the diagonal).

[1] T. H. Cormen, C. E. Leiserson and R. L. Rivest (1999). *Introduction to algorithms*, The MIT Press, Cambridge, Massachusetts.

[2] R. E. Tarjan (1972). *Depth-first search and linear graph algorithms*, SIAM J. Comput., **1**, 146-160.

# 5 EXAMPLE OF USE

## 5.1 First example: find a matching

In our first example, we give the code required to generate a matching using HSL_MC79. We generate a matching for an indefinite matrix with the following sparsity structure:

$$\begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & x \\ x & & x & \end{pmatrix} \tag{5.1}$$

The following code may be used

```
program main
  use hsl_mc79_integer
  implicit none

  ! Local variables
  integer :: m, n, ne

  integer, dimension (:), allocatable :: row, ptr, rowmatch, colmatch

  type (mc79_control) :: control
  type (mc79_info) :: info

  ! Read in the number of rows, the number of columns, and the number
  ! of non-zeros in the matrix
  read (5,*) m, n, ne

  ! Allocate arrays
  allocate (row(ne),ptr(n+1),rowmatch(m),colmatch(n))

  ! Read in pointers for the matrix
  read (5,*) ptr(1:n+1)

  ! Read in row indices for the matrix
  read (5,*) row(1:ne)

  ! Find matching
  call mc79_matching(m,n,ptr,row,rowmatch,colmatch,control,info)
  write(6,'(a15)') 'rowmatch ='
  write(6,'(5i15)') rowmatch
  write(6,'(a15)') 'colmatch ='
  write(6,'(5i15)') colmatch
  write(6,'(a15)') 'info%mbar ='
  write(6,'(5i15)') info%mbar
  write(6,'(a15)') 'info%nbar ='
  write(6,'(5i15)') info%nbar

  ! Deallocate arrays
  deallocate (row,ptr,rowmatch,colmatch)

end program main
```

with the following data:

```
8 7 9
1 2 4 5 6 8 9 10
1 2 5 7 3 1 4 8 3
```

This produces the following output:

```
     rowmatch =
               1              2              4              5              0
               0              3              6
     colmatch =
               1              2              7              3              4
               8              0
    info%mbar =
               2
    info%nbar =
               1
```

### 5.2 Second Example: Coarse Dulmage-Mendelsohn Decomposition

In our second example, we give the code required to generate a coarse Dulmage-Mendelsohn decomposition for (5.1) using HSL_MC79. The following code may be used

```
program main
  use hsl_mc79_integer
  implicit none

  ! Local variables
  integer :: m, n, ne

  integer, dimension (:), allocatable :: row, ptr, perm, rowperm, colperm

  type (mc79_control) :: control
  type (mc79_info) :: info

  ! Read in the number of rows, the number of columns, and the number
  ! of non-zeros in the matrix
  read (5,*) m, n, ne

  ! Allocate arrays
  allocate (row(ne),ptr(n+1),rowperm(m),colperm(n))

  ! Read in pointers for the matrix
  read (5,*) ptr(1:n+1)

  ! Read in row indices for the matrix
  read (5,*) row(1:ne)

  ! Find coarse Dulmage-Mendelsohn decomposition
  call mc79_coarse(m,n,ptr,row,rowperm,colperm,control,info)
  write(6,'(a15)') 'rowperm ='
  write(6,'(5i15)') rowperm
  write(6,'(a15)') 'colperm ='
  write(6,'(5i15)') colperm
  write(6,'(a15)') 'info%m1 ='
  write(6,'(5i15)') info%m1
  write(6,'(a15)') 'info%m2 ='
  write(6,'(5i15)') info%m2
  write(6,'(a15)') 'info%m3 ='
  write(6,'(5i15)') info%m3
  write(6,'(a15)') 'info%n1 ='
  write(6,'(5i15)') info%n1
  write(6,'(a15)') 'info%n2 ='
  write(6,'(5i15)') info%n2
  write(6,'(a15)') 'info%n3 ='
  write(6,'(5i15)') info%n3

  ! Deallocate arrays
  deallocate (row,ptr,rowperm,colperm)

end program main
```

with the following data:

```
8 7 9
1 2 4 5 6 8 9 10
1 2 5 7 3 1 4 8 3
```

This produces the following output:

```
      rowperm =
              3              1              4              7              8
              2              6              5
      colperm =
              7              4              1              3              5
              6              2
      info%m1 =
              1
      info%m2 =
              4
      info%m3 =
              3
      info%n1 =
              2
      info%n2 =
              4
      info%n3 =
              1
```

### 5.3 Third Example: Fine Dulmage-Mendelsohn Decomposition

In our third example, we give the code required to generate a fine Dulmage-Mendelsohn decomposition for (5.1) using HSL_MC79. The following code may be used

```
program main
  use hsl_mc79_integer
  implicit none

  ! Local variables
  integer :: m, n, ne

  integer, dimension (:), allocatable :: row, ptr, perm, rowperm, colperm, &
       rowptr, colptr

  type (mc79_control) :: control
  type (mc79_info) :: info

  ! Read in the number of rows, the number of columns, and the number
  ! of non-zeros in the matrix
  read (5,*) m, n, ne

  ! Allocate arrays
  allocate (row(ne),ptr(n+1),rowperm(m),colperm(n),rowptr(m+2),colptr(n+2))

  ! Read in pointers for the matrix
  read (5,*) ptr(1:n+1)

  ! Read in row indices for the matrix
  read (5,*) row(1:ne)

  ! Find coarse Dulmage-Mendelsohn decomposition
  call mc79_fine(m,n,ptr,row,rowperm,colperm,rowptr,colptr,control,info)
  write(6,'(a)') 'Results from mc79_fine'
```

```
  write(6,'(a15)') 'rowperm ='
  write(6,'(5i15)') rowperm
  write(6,'(a15)') 'colperm ='
  write(6,'(5i15)') colperm
  write(6,'(a15)') 'rowptr ='
  write(6,'(5i15)') rowptr
  write(6,'(a15)') 'colptr ='
  write(6,'(5i15)') colptr
  write(6,'(a15)') 'info%hz_comps ='
  write(6,'(5i15)') info%hz_comps
  write(6,'(a15)') 'info%sq_comps ='
  write(6,'(5i15)') info%sq_comps
  write(6,'(a15)') 'info%vt_comps ='
  write(6,'(5i15)') info%vt_comps

  ! Deallocate arrays
  deallocate (row,ptr,rowperm,colperm,rowptr,colptr)

end program main
```

with the following data:

```
8 7 9
1 2 4 5 6 8 9 10
1 2 5 7 3 1 4 8 3
```

This produces the following output:

```
Results from mc79_fine
      rowperm =
              3              1              4              7              8
              2              5              6
      colperm =
              4              7              1              5              3
              6              2
       rowptr =
              1              2              3              4              5
              6              9              0              0              0
       colptr =
              1              3              4              5              6
              7              8              0              0
info%hz_comps =
              1
info%sq_comps =
              4
info%vt_comps =
              1
```

### 5.4 Fourth Example: matching with coordinate input

In our final example, we give the code required to generate a matching for (5.1) using HSL_MC79 when the matrix data is stored in coordinate format. We use HSL_MC69 to convert the input to standard HSL format. The following code may be used

```
program main
  use hsl_mc79_integer
  use hsl_mc69_double
  implicit none
```

```
  ! Local variables
  integer :: m, n, ne_in, ne, flag, matrix_type

  integer, dimension (:), allocatable :: row_in, col_in, row, ptr, &
        rowmatch, colmatch

  type (mc79_control) :: control
  type (mc79_info) :: info

  ! Read in the number of rows, the number of columns, and the number
  ! of non-zeros in the matrix
  read (5,*) m, n, ne_in

  ! Allocate arrays
  ALLOCATE (row_in(ne_in),col_in(ne_in),ptr(n+1),rowmatch(m),colmatch(n))

  ! Read in row indices for the matrix
  READ (5,*) row_in(1:ne_in)

  ! Read in column indices for the matrix
  READ (5,*) col_in(1:ne_in)

  ! Convert matrix into standard HSL format
  matrix_type = 1
  CALL mc69_coord_convert(matrix_type,m,n,ne_in,row_in,col_in,ptr,row,flag)
  WRITE (6,'(a15)') 'ptr ='
  WRITE (6,'(5i15)') ptr
  WRITE (6,'(a15)') 'row ='
  WRITE (6,'(5i15)') row

  ! Find matching
  call mc79_matching(m,n,ptr,row,rowmatch,colmatch,control,info)
  write(6,'(a15)') 'rowmatch ='
  write(6,'(5i15)') rowmatch
  write(6,'(a15)') 'colmatch ='
  write(6,'(5i15)') colmatch
  write(6,'(a15)') 'info%mbar ='
  write(6,'(5i15)') info%mbar
  write(6,'(a15)') 'info%nbar ='
  write(6,'(5i15)') info%nbar

  ! Deallocate arrays
  deallocate (row_in,col_in,row,ptr,rowmatch,colmatch)

end program main
```

with the following data:

```
8 7 9
1 2 5 7 3 1 4 8 3
1 2 2 3 4 5 5 6 7
```

This produces the following output:

```
          ptr =
              1              2              4              5              6
              8              9             10
          row =
              1              2              5              7              3
              1              4              8              3
    rowmatch =
              1              2              4              5              0
              0              3              6
    colmatch =
```

```
           1             2             7             3             4
           8             0
 info%mbar =
           2
 info%nbar =
           1
```