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 C interface to Fortran code

This package is written in Fortran and a wrapper is provided for C programmers. This wrapper may only implement a subset of the full functionality described in the Fortran user documentation.

All subroutines will automatically convert between 0-based (C) and 1-based (Fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing. The conversion may be disabled by setting the control

parameter control.f_arrays≠0 (i.e. true) and supplying all data using 1-based indexing. With 0-based indexing, the matrix is treated as having rows $0, 1, \ldots, m-1$ and columns $0, 1, \ldots, n-1$. In this document, we assume 0-based indexing when referencing the matrix.

The wrapper uses the Fortran 2003 interoperability features. **Matching C and Fortran compilers must be used**, for example, gcc and gfortran, or icc and ifort. If the Fortran compiler is not used to link the user's program, additional Fortran compiler libraries may need to be linked explicitly.

2.2 Calling sequences

Access to the package requires inclusion of a header file

```
#include "hsl_mc79i.h"
```

The following subroutines are available to the user:

(a) To initialise members of struct mc79_control to their default values, mc79_default_control may be called.

(b) To compute a maximum matching, mc79_matching should be called.

(c) To compute a coarse Dulmage-Mendelsohn decomposition, mc79_coarse should be called.

(d) 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 2.4.1.

2.3 The derived data types

The user must employ the structure defined in the header file to declare scalars of type mc79_control and mc79_info. The following pseudocode illustrates this.

```
#include "hsl_mc79i.h"
...
struct mc79_control control;
struct mc79_info info;
```

The members of mc79_control and mc79_info are described in Section 2.5.1 and Section 2.5.2, respectively.

2.4 Argument lists and calling sequences

2.4.1 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.

2.4.2 The default setting subroutine

Default values for members of the mc79_control structure may be set by a call to mc79_default_control.

```
void mc79_default_control(struct mc79_control *control);
```

control has its members set to their default values, as described in Section 2.5.1.

2.4.3 To compute the maximum matching

The method constructs a maximum matching for the matrix A.

```
void mc79_matching(int m, int n, const int ptr[], const int row[],
   int rowmatch[], int colmatch[], const struct mc79_control *control,
   struct mc79_info *info);
```

m must hold the number of rows in A. **Restriction:** m≥ 1.

n must hold the number of columns in A. **Restriction:** n≥ 1.

ptr is a rank-one array of 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] must be set to one more than the total number of entries in A.

row is a rank-one array of size ptr[n]-1. The entries must hold the row indices of the entries of A, with the row indices for the entries in column 0 preceding those for column 1, and so on.

rowmatch is a rank-one array of size m. On exit, if rowmatch[i]=-1 (or 0 if control.f_arrays$\neq$0), then row i of A is not matched to any column. If rowmatch[i]=j and j$\geq$0, then row i of A is matched to column j of A.

colmatch is a rank-one array of size n. On exit, if colmatch[j]=-1 (or 0 if control.f_arrays$\neq$0), then column j of A is not matched to any column. If colmatch[j]=i and i$\geq$0, then column j of A is matched to row i of A.

control is used to control the action, as explained in Section 2.5.1.

info is used to provide information about the execution of the subroutine, as explained in Section 2.5.2. In particular, info.flag is used as an error/warning flag, as detailed in Section 2.6.

2.4.4 To compute a coarse Dulmage-Mendelsohn decomposition

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

```
void mc79_coarse(int m, int n, const int ptr[], const int row[],
   int rowperm[], int colperm[], const struct mc79_control *control,
   struct mc79_info *info);
```

m, n, ptr, row, control and info are all as in the call to mc79_matching.

rowperm is a rank-one array of size m. On exit, if rowperm[i]=j, then row j of A becomes row i of the permuted matrix.

colperm is a rank-one array of size n. On exit, if colperm[i]=j, then column j of A becomes column i of the permuted matrix.

2.4.5 To compute a fine Dulmage-Mendelsohn decomposition

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

```
void mc79_fine(int m, int n, const int ptr[], const int row[],
      int rowperm[], int colperm[], int rowcomp[], int colcomp[],
      const struct mc79_control *control, struct mc79_info *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 a rank-one array of size `m+2`. On exit, it holds the index in the reordered matrix of the first row in each component:

`rowptr[0]`,..., `rowptr[info.hz_comps-1]` 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]`,...,`rowptr[info.hz_comps+info.sq_comps-1]` 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]`,...,`rowptr[info.hz_comps+info.sq_comps+info.vt_comps-1]` 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]` is equal to `m+1`. The remaining entries are set to `0`.

`colptr` is a rank-one array of size `n+2`. On exit, it holds the index in the reordered matrix of the first column in each component:

`colptr[0]`,...,`colptr[info.hz_comps-1]` 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]`,...,`colptr[info.hz_comps+info.sq_comps-1]` 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]`,...,`colptr[info.hz_comps+info.sq_comps+info.vt_comps-1]` 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]` is equal to `n+1`. The remaining entries are set to `0`.

2.5 The derived data types

2.5.1 The derived data type for holding control parameters

The derived data type `mc79_control` is used to hold controlling data. The members, which may be given default values through a call to `mc79_default_control`, are:

C only controls

int f_arrays indicates whether to use C or Fortran array indexing. If f_arrays≠0 (i.e. evaluates to true) then 1-based indexing of the arrays ptr, row, rowperm, colperm, rowmatch, colmatch, rowptr and colptr is assumed. Otherwise, if f_arrays=0 (i.e. evaluates to false), then these arrays are copied and converted to 1-based indexing in the wrapper function. All descriptions in this documentation assume f_arrays=0. The default is f_arrays=0 (false).

Printing controls

int lp specifies the Fortran unit number for error messages. If it is negative, these messages will be suppressed. The default value is 6 (stdout).

int mp specifies the Fortran unit number for diagnostic messages. If it is negative, these messages will be suppressed. The default value is 6 (stdout).

print_level specifies 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.

2.5.2 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

int flag gives the exit status of the algorithm (details in Section 2.6).

int mbar holds the number of rows that cannot be matched to columns in a maximum matching of A.

int nbar holds the number of columns that cannot be matched to rows in a maximum matching of A.

int stat holds the Fortran stat parameter.

Information returned by mc79_coarse and mc79_fine only

int m1 holds the number of rows in the submatrix A_1, where A_1 is defined by equation (1.1).

int m2 holds the number of rows in the submatrix A_2, where A_2 is defined by equation (1.1).

int m3 holds the number of rows in the submatrix A_3, where A_3 is defined by equation (1.1).

int n1 holds the number of columns in the submatrix A_1, where A_1 is defined by equation (1.1).

int n2 holds the number of columns in the submatrix A_2, where A_2 is defined by equation (1.1).

int n3 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

int hz_comps holds the number of components found in A_1, where A_1 is defined by equation (1.1).

int sq_comps holds the number of components found in A_2, where A_2 is defined by equation (1.1).

int vt_comps holds the number of components found in A_3, where A_3 is defined by equation (1.1).

2.6 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 Fortran unit `control.lp`.
Possible negative values are:

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

`-2` Memory deallocation failed. If available, the Fortran `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 2003 subset (F95 + TR15581 + C interoperability).

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

```
/* hsl_mc79is.c */
/* Simple code to demonstrate finding a matching with HSL_MC79 */

#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc79i.h"

int main(void) {
   int i, m, n, ne;
   int *row, *ptr, *rowmatch, *colmatch;
   struct mc79_control control;
   struct mc79_info info;

   /* Read in the number of rows, the number of columns, and the number
    * of non-zeros in the matrix */
   scanf("%d %d %d", &m, &n, &ne);

   /* Allocate arrays */
   ptr = (int *) malloc((n+1)*sizeof(int));
   row = (int *) malloc(ne*sizeof(int));
   rowmatch = (int *) malloc(m*sizeof(int));
   colmatch = (int *) malloc(n*sizeof(int));

   /* Read in pointers for the matrix */
```

```
   for(i=0; i<n+1; i++) scanf("%d", &(ptr[i]));

   /* Read in row indices for the matrix */
   for(i=0; i<ne; i++) scanf("%d", &(row[i]));

   /* Initialise control structure */
   mc79_default_control(&control);

   /* Find matching */
   mc79_matching(m, n, ptr, row, rowmatch, colmatch, &control, &info);
   if(info.flag < 0) {
      printf("Error return from mc79_matching. info.flag = %d\n", info.flag);
      free(ptr); free(row); free(rowmatch); free(colmatch);
      return 1;
   }

   /* Print out results */
   printf("rowmatch = \n");
   for(i=0; i<m; i++) printf(" %d", rowmatch[i]);
   printf("\ncolmatch = \n");
   for(i=0; i<n; i++) printf(" %d", colmatch[i]);
   printf("\ninfo.mbar = %d\n", info.mbar);
   printf("info.nbar = %d\n", info.nbar);

   /* Deallocate arrays */
   free(ptr); free(row); free(rowmatch); free(colmatch);

   return 0;
}
```

with the following data:

```
8 7 9
0 1 3 4 5 7 8 9
0 1 4 6 2 0 3 7 2
```

This produces the following output:

```
rowmatch =
 0 1 3 4 -1 -1 2 5
colmatch =
 0 1 6 2 3 7 -1
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

```
/* hsl_mc79is1.c */
/* Simple code to demonstrate finding a coarse decomposition with HSL_MC79 */

#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc79i.h"

int main(void) {
  int i, m, n, ne;
  int *row, *ptr, *rowperm, *colperm;
```

```
  struct mc79_control control;
  struct mc79_info info;

   /* Read in the number of rows, the number of columns, and the number
    * of non-zeros in the matrix */
   scanf("%d %d %d", &m, &n, &ne);

   /* Allocate arrays */
   ptr = (int *) malloc((n+1)*sizeof(int));
   row = (int *) malloc(ne*sizeof(int));
   rowperm = (int *) malloc(m*sizeof(int));
   colperm = (int *) malloc(n*sizeof(int));

   /* Read in pointers for the matrix */
   for(i=0; i<n+1; i++) scanf("%d", &(ptr[i]));

   /* Read in row indices for the matrix */
   for(i=0; i<ne; i++) scanf("%d", &(row[i]));

   /* Initialise control structure */
   mc79_default_control(&control);

   /* Find coarse Dulmage-Mendelsohn decomposition */
   mc79_coarse(m, n, ptr, row, rowperm, colperm, &control, &info);
   if(info.flag < 0) {
      printf("Error return from mc79_coarse. info.flag = %d\n", info.flag);
      free(ptr); free(row); free(rowperm); free(colperm);
      return 1;
   }

   /* Print out results */
   printf("rowperm = \n");
   for(i=0; i<m; i++) printf(" %d", rowperm[i]);
   printf("\ncolperm = \n");
   for(i=0; i<n; i++) printf(" %d", colperm[i]);
   printf("\ninfo.m1 = %d\n", info.m1);
   printf("info.m2 = %d\n", info.m2);
   printf("info.m3 = %d\n", info.m3);
   printf("info.n1 = %d\n", info.n1);
   printf("info.n2 = %d\n", info.n2);
   printf("info.n3 = %d\n", info.n3);

   /* Deallocate arrays */
   free(ptr); free(row); free(rowperm); free(colperm);

   return 0;
}
```

with the following data:

```
8 7 9
0 1 3 4 5 7 8 9
0 1 4 6 2 0 3 7 2
```

This produces the following output:

```
rowperm =
 2 0 3 6 7 1 5 4
colperm =
 6 3 0 2 4 5 1
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

```
/* hsl_mc79is2.c */
/* Simple code to demonstrate finding a fine decomposition with HSL_MC79 */

#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc79i.h"

int main(void) {
  int i, m, n, ne;
  int *row, *ptr, *rowperm, *colperm, *rowptr, *colptr;
  struct mc79_control control;
  struct mc79_info info;

   /* Read in the number of rows, the number of columns, and the number
    * of non-zeros in the matrix */
   scanf("%d %d %d", &m, &n, &ne);

   /* Allocate arrays */
   ptr = (int *) malloc((n+1)*sizeof(int));
   row = (int *) malloc(ne*sizeof(int));
   rowperm = (int *) malloc(m*sizeof(int));
   colperm = (int *) malloc(n*sizeof(int));
   rowptr = (int *) malloc((m+2)*sizeof(int));
   colptr = (int *) malloc((n+2)*sizeof(int));

   /* Read in pointers for the matrix */
   for(i=0; i<n+1; i++) scanf("%d", &(ptr[i]));

   /* Read in row indices for the matrix */
   for(i=0; i<ne; i++) scanf("%d", &(row[i]));

   /* Initialise control structure */
   mc79_default_control(&control);

   /* Find fine Dulmage-Mendelsohn decomposition */
   mc79_fine(m, n, ptr, row, rowperm, colperm, rowptr, colptr,
         &control, &info);
   if(info.flag < 0) {
      printf("Error return from mc79_fine. info.flag = %d\n", info.flag);
      free(ptr); free(row); free(rowperm); free(colperm);
      free(rowptr); free(colptr);
      return 1;
   }

   /* Print out results */
   printf("Results from mc79_fine\n");
   printf("rowperm = \n");
   for(i=0; i<m; i++) printf(" %d", rowperm[i]);
   printf("\ncolperm = \n");
   for(i=0; i<n; i++) printf(" %d", colperm[i]);
   printf("\nrowptr = \n");
   for(i=0; i<m+2; i++) printf(" %d", rowptr[i]);
   printf("\ncolptr = \n");
   for(i=0; i<n+2; i++) printf(" %d", colptr[i]);
   printf("\ninfo.hz_comps = %d\n", info.hz_comps);
   printf("info.sq_comps = %d\n", info.sq_comps);
   printf("info.vt_comps = %d\n", info.vt_comps);

   /* Deallocate arrays */
   free(ptr); free(row); free(rowperm); free(colperm);
   free(rowptr); free(colptr);
```

```
   return 0;
}
```

with the following data:

```
8 7 9
0 1 3 4 5 7 8 9
0 1 4 6 2 0 3 7 2
```

This produces the following output:

```
Results from mc79_fine
rowperm =
 2 0 3 6 7 1 4 5
colperm =
 3 6 0 4 2 5 1
rowptr =
 0 1 2 3 4 5 8 -1 -1 -1
colptr =
 0 2 3 4 5 6 7 -1 -1
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 initially stored in coordinate format. We use HSL_MC69 to convert the input to standard HSL format. The following code may be used

```
/* hsl_mc79is3.c */
/* Simple code to demonstrate finding a matching with HSL_MC79 but with
 * matrix data initially in coordinate format */

#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc69d.h"
#include "hsl_mc79i.h"

int main(void) {
   int i, m, n, ne_in, flag, lrow;
   int *row_in, *col_in, *row, *ptr, *rowmatch, *colmatch;
   struct mc79_control control;
   struct mc79_info info;

   /* Read in the number of rows, the number of columns, and the number
    * of non-zeros in the matrix */
   scanf("%d %d %d", &m, &n, &ne_in);

   /* Allocate arrays */
   row_in = (int *) malloc(ne_in*sizeof(int));
   col_in = (int *) malloc(ne_in*sizeof(int));
   rowmatch = (int *) malloc(m*sizeof(int));
   colmatch = (int *) malloc(n*sizeof(int));

   /* Read in row indices for the matrix */
   for(i=0; i<ne_in; i++) scanf("%d", &(row_in[i]));
```

```
   /* Read in column indices for the matrix */
   for(i=0; i<ne_in; i++) scanf("%d", &(col_in[i]));

   /* Convert matrix into standard HSL format */
   ptr = (int *) malloc((n+1)*sizeof(int));
   lrow = ne_in; /* maximum size of output cannot exceed size of input */
   row = (int *) malloc(lrow*sizeof(int));
   flag = mc69_coord_convert(-1, HSL_MATRIX_REAL_RECT, 0, m, n, ne_in, row_in,
         col_in, NULL, ptr, lrow, row, NULL, NULL, NULL, NULL, NULL);
   if(flag < 0) {
      printf("Error return from mc69_coord_convert. flag = %d\n", flag);
      free(row_in); free(col_in);
      free(ptr); free(row); free(rowmatch); free(colmatch);
      return 1;
   }
   printf("ptr = \n");
   for(i=0; i<n+1; i++) printf(" %d", ptr[i]);
   printf("\nrow = \n");
   for(i=0; i<ptr[n]; i++) printf(" %d", row[i]);
   printf("\n");

   /* Initialise control structure */
   mc79_default_control(&control);

   /* Find matching */
   mc79_matching(m, n, ptr, row, rowmatch, colmatch, &control, &info);
   if(info.flag < 0) {
      printf("Error return from mc79_matching. info.flag = %d\n", info.flag);
      free(row_in); free(col_in);
      free(ptr); free(row); free(rowmatch); free(colmatch);
      return 1;
   }

   /* Print out results */
   printf("rowmatch = \n");
   for(i=0; i<m; i++) printf(" %d", rowmatch[i]);
   printf("\ncolmatch = \n");
   for(i=0; i<n; i++) printf(" %d", colmatch[i]);
   printf("\ninfo.mbar = %d\n", info.mbar);
   printf("info.nbar = %d\n", info.nbar);

   /* Deallocate arrays */
   free(row_in); free(col_in);
   free(ptr); free(row); free(rowmatch); free(colmatch);

   return 0;
}
```

with the following data:

```
8 7 9
0 1 4 6 2 0 3 7 2
0 1 1 2 3 4 4 5 6
```

This produces the following output:

```
ptr =
 0 1 3 4 5 7 8 9
row =
 0 1 4 6 2 0 3 7 2
rowmatch =
 0 1 3 4 -1 -1 2 5
colmatch =
 0 1 6 2 3 7 -1
info.mbar = 2
info.nbar = 1
```