Science and Technology Facilities Council

HSL_MC56

PACKAGE SPECIFICATION HSL

1 SUMMARY

To **read a file containing a sparse matrix held in Rutherford-Boeing format and manipulate it to the desired sparse matrix format**. The user may specify the storage format the matrix is returned in, and random values can be optionally generated to match the matrix pattern. Rutherford-Boeing format can be used to specify either matrix data or supplementary data, however this package only supports matrix data. To read supplementary data, the HSL routine `MC56` can be used. To write data in Rutherford-Boeing format the HSL routines `MC54` and `MC55` can be used for matrix data and supplementary data, respectively.

For information on the Rutherford-Boeing sparse matrix format, the user should consult the report

[1] I.S. Duff, R.G. Grimes and J.G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Rutherford Appleton Laboratory Technical Report RAL-TR-97-031 Revision 1, 1999.

ATTRIBUTES — Version: 1.1.0. **Types:** Real (single, double), Integer, Complex (single, double). **Calls:** `HSL_FA14`, `HSL_MC34`, `MC56`, `HSL_ZD11`, `HSL_ZD13`. **Date:** April 2010. **Origin:** J. D. Hogg, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components).

2 HOW TO USE THE PACKAGE

Access to the package requires a `USE` statement such as

Single precision version

```
USE HSL_MC56_single
```

Double precision version

```
USE HSL_MC56_double
```

Integer version

```
USE HSL_MC56_integer
```

Complex version

```
USE HSL_MC56_complex
```

Double complex version

```
USE HSL_MC56_double_complex
```

The version used determines the type of the values returned. Conversion is automatically performed from the file type if it is compatible, otherwise an error is returned. File type is determined from a three-letter type code in the header of the Rutherford-Boing file. For matrices, the first letter of this type code indicates the data type and is one of the following:

`"p"`, `"q"` - the file contains only the pattern of the matrix (`"q"` indicates the values are in an supplementary file). This can be read by any version of `HSL_MC56`.

`"i"` - the file contains an integer matrix. This can be read by any version of `HSL_MC56`.

`"r"` - the file contains a real matrix. This can be read by the `_single`, `_double`, `_complex` and `_double_complex` versions of `HSL_MC56`.

`"c"` - the file contains a complex matrix. This can be read by the `_complex` and `_double_complex` versions of `HSL_MC56`.

If it is required to use more than one of these modules at the same time, the derived type mc56_control described in Section 2.1 and the subroutine mc56_peek should be used only from one of the packages. This can be done, for example, as follows:

```
use hsl_mc56_double
use hsl_mc56_double_complex, only: mc56_read
```

The following subroutines are available to the user:

- mc56_peek may be called to read the header information of a file. This may be used determine the file type code so that the correct version of HSL_MC56 may be used on a subsequent call to mc56_read.
- mc56_read reads a matrix held in Rutherford-Boeing format into the supplied zd11_type or zd13_type variable. Optionally, the matrix may be converted between standard sparse matrix formats and random values may be generated to match a sparsity pattern read from the file.

2.1 The derived data types

For each problem, the user must employ the derived type defined by the module to declare a scalar of type mc56_control. The following pseudocode illustrates this.

```
use hsl_mc56_double
...
type (zd11_type) :: matrix
type (mc56_control) :: control
```

The derived data type zd11_type is used to return a sparse matrix in assembled form. To read an sparse matrix stored by elements, the derived data type zd13_type is used instead. The components of mc56_control are described in Section 2.2.4. The various sparse matrix formats offered by this package are described in Section 4.

2.2 Argument lists and calling sequences

2.2.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments, which are always at the end of the argument list. Since we reserve the right to modify the argument list and to add additional optional arguments in future releases of the code, **we strongly recommend that all optional arguments be called by keyword, not by position**.

2.2.2 To obtain information about a matrix in a Rutherford-Boeing file

To obtain information found in the header of a file, the user may call the routine

```
call mc56_peek(filename, info[, m, n, nelt, nvar, nval, type_code, title, identifier])
```

filename is a scalar INTENT(IN) argument of type CHARACTER(LEN=*). It specifies the filename of the file to be read. HSL_MC56 will open the file, read the header and then close the file.

info is a scalar INTENT(OUT) argument of type INTEGER. On exit it contains a status code indicating success or failure of the subroutine, see Section 2.3 for details.

m is an optional scalar INTENT(OUT) argument of type INTEGER. On exit it contains the number of rows in the matrix.

n is an optional scalar INTENT(OUT) argument of type INTEGER. On exit it contains the number of columns in the matrix.

nelt is an optional scalar INTENT(OUT) argument of type INTEGER. On exit it contains the number of elements in the file, or 0 if the matrix is assembled.

nvar is an optional scalar INTEGER(OUT) argument of type INTEGER. On exit it contains the number of variable indices specified in the file.

nval is an optional scalar INTEGER(OUT) argument of type INTEGER. On exit it contains the number of values specified in the file.

type_code is an optional INTENT(OUT) argument of type CHARACTER(LEN=3). On exit it contains the three-letter type code found in the Rutherford-Boeing file.

title is an optional INTENT(OUT) argument of type CHARACTER(LEN=72). On exit it contains the title of the matrix read from the Rutherford-Boeing file.

identifier is an optional INTENT(OUT) argument of type CHARACTER(LEN=8). On exit it contains the 8 character identifier read from the Rutherford-Boeing file.

2.2.3 To read a matrix from a Rutherford-Boeing file

To read a file, the user may call the routine

```
call mc56_read(filename, matrix, control, info, [type_code, title, identifier, seed])
```

filename is a scalar INTENT(IN) argument of type CHARACTER(LEN=*). It specifies the filename of the file to be read. HSL_MC56 will open the file, read it and close it.

matrix is a scalar INTENT(OUT) argument of type ZD11_type if the matrix to be read is an assembled matrix, or ZD12_type if the matrix to be read is in element form.

control is a scalar INTENT(IN) argument of type MC56_control that contains parameters affecting how the file is interpreted. See Section 2.2.4 for further details.

info is a scalar INTENT(OUT) argument of type INTEGER. On exit it contains a status code indicating success or failure of the subroutine, see Section 2.3 for details.

type_code is an optional INTENT(OUT) argument of type CHARACTER(LEN=3). On exit it contains the three letter type code found in the Rutherford-Boeing file.

title is an optional INTENT(OUT) argument of type CHARACTER(LEN=72). On exit it contains the title of the matrix read from the Rutherford-Boeing file.

identifier is an optional INTENT(OUT) argument of type CHARACTER(LEN=8). On exit it contains the 8 character identifier read from the Rutherford-Boeing file.

seed is an optional INTENT(INOUT) argument of type FA14_seed. If present and random values are to be generated (control%values $\geq$ 2), it is used as the seed for the random number generator used. If it is not present, HSL_MC56 will use its own random seed.

2.2.4 The derived data type for holding control parameters

The derived data type `MC56_control` is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are:
*Components that apply to both **assembled** and **elemental** entry:*

`extra_space` is a scalar of type default `REAL`. When allocating array components `matrix`, the amount of space allocated will be the minimum amount required multiplied by this number. The default value is `1.0`.

`lwr_upr_full` is a scalar of type `INTEGER`. It is only used when the matrix to be read is symmetric, skew symmetric or Hermitian, and it can take the following values:

`1` Only lower triangular entries are returned.

`2` Only upper triangular entries are returned.

`3` Both lower and upper triangular entries are returned.

The default value is `1`.

*Components that apply to only **assembled** entry:*

`add_diagonal` is a logical of type `LOGICAL`. If `.true.`, then missing diagonal entries will be added to the matrix as explicit zeros.

`format` is a scalar of type `INTEGER`. It determines the output sparse matrix format for assembled matrices, and can take the following values:

`1` Compressed sparse column. `matrix%ptr` and `matrix%row` are used to store the pattern, and `matrix%val` to store the values if required.

`2` Compressed sparse row. `matrix%ptr` and `matrix%col` are used to store the pattern, and `matrix%val` to store the values if required.

`3` Coordinate. `matrix%row` and `matrix%col` are used to store the pattern, and `matrix%val` to store the values if required.

The default value is `1`.

`values` is a scalar of type `INTEGER`. It determines what, if any, values are returned. It can take the values listed below. If a negative value is supplied then values from the file are ignored, if positive then values are only generated if no values are found in the file.

`0` Matrix values are returned if and only if they are given in the file to be read.

`1` Matrix values are not returned, only the pattern is read.

`-2,2` Values are distributed uniformally over the interval $[-1,1]$. If the type code indicates the pattern is symmetric, skew-symmetric or Hermitian, this is reflected in the numerically values.

`-3,3` If the matrix is real symmetric or complex Hermitian, the off-diagonal values are distributed uniformally over the interval $[-1,1]$ (real) or ball $\|x\|_\infty < 1.0$ (complex) such that the matrix has the specified property. Diagonal entries are given a real value $\max(100.0, 10k)$, where k is the number of entries in the column. This ensures the matrix is strongly diagonally dominant, and hence that a Cholesky factorization exists. Additional entries are added to the diagonal if required. If the matrix is not real symmetric or complex Hermitian, this option is equivalent to `control%values = 2`.

`-4,4` If the matrix is symmetric, skew symmetric or Hermitian, but `control%lwr_upr_full = 3`, values are distributed uniformally over the interval $[-1,1]$ and the matrix is numerically unsymmetric. If the matrix is unsymmetric, or `control%lwr_upr_full` $\neq$ `3`, this has the same effect as `control%values = 2`.

The default value is `0`.

2.3 Warning and error messages

A successful return is indicated by `info` having the value zero. A negative value is associated with an error. Possible negative values are:

`-1` Unable to find an available unit on which to open the file.

`-2` File does not exist, or insufficient permissions to read the file.

`-3` File does not appear to be in Rutherford-Boeing format.

`-4` Error on file i/o.

`-5` Attempted to read non-compatible type. Complex matrices cannot be read with the `_integer`, `_single` or `_double` versions of this package. Real matrices cannot be read with the `_integer` version of this package.

`-6` Attempted to read elemental matrix into assembled data type, or assembled matrix into elemental data type.

`-10` `control%extra_space < 1.0`.

`-11` `control%lwr_upr_full` has invalid value.

`-12` `control%format` has invalid value.

`-13` `control%values` has invalid value.

`-20` Allocation error.

Possible positive values are:

`+1` Auxiliary file exists. The first letter of `type_code` read from the file is ‘`q`’. This indicates that there exists an auxiliary file containing matrix values, however only the pattern has been read. Values to match the pattern found have been generated if `control%values` < 0 or ≥ 2.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: `HSL_FA14`, `HSL_MC34`, `MC56`, `HSL_ZD11`, `HSL_ZD13`.

Input/output: No printing or other output occurs. The specified file in opened read-only mode on a unit number chosen by `HSL_MC56`, is read and is then closed before returning to the user.

Restrictions: `filename` must be a valid Rutherford-Boing file, `control%extra_space` ≥ 1.0,
$1 \leq$ `control%lwr_upr_full` ≤ 3, $1 \leq$ `control%format` ≤ 3, `control%values` $= -4, -3, -2, 0, 1, 2, 3,$ or 4.

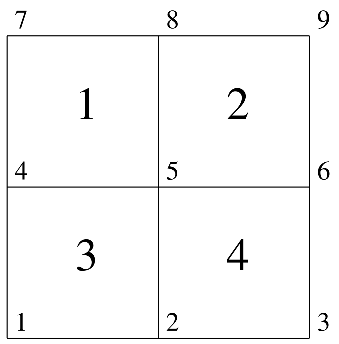
Portability: Fortran 95, plus allocatable components of derived types.

4 METHOD

This package provides a wrapper around different versions of the original Fortran 77 package `MC56`, using `HSL_MC34` for the expansion of symmetric, skew symmetric and Hermitian matrices from lower triangular to full storage and `HSL_FA14` for the generation of random values. Additional code was written in this package to swap between lower and upper representations of such matrices.

In the following subsection we describe how data is returned from this package in various sparse matrix formats at the request of the user. The array sizes described apply only to the case where it is not necessary for the package to manipulate the data, for example to add diagonal entries.

We will consider the following example, derived from a simple finite element problem. Consider the below mesh comprising four 4-noded quadrilateral elements with one degree of freedom at each node $i, 1 \leq i \leq 6$ (the nodes 7, 8, and 9 are assumed constrained).



The four element matrices $A^{[k]}$ $(1 \leq k \leq 4)$ are:

$$\begin{array}{c} 4 \\ 5 \end{array} \begin{pmatrix} 2. & 1. \\ -1. & 7. \end{pmatrix} \qquad \begin{array}{c} 5 \\ 6 \end{array} \begin{pmatrix} 3. & 2. \\ 4. & 8. \end{pmatrix} \qquad \begin{array}{c} 4 \\ 5 \\ 1 \\ 2 \end{array} \begin{pmatrix} 4. & 3. & 2. & 3. \\ -1. & 1. & 3. & 2. \\ 2. & 3. & 6. & 1. \\ 3. & 2. & 1. & 5. \end{pmatrix} \qquad \begin{array}{c} 5 \\ 6 \\ 2 \\ 3 \end{array} \begin{pmatrix} 2. & 1. & 8. & 3. \\ 1. & 3. & 2. & 2. \\ 8. & 2. & 2. & 5. \\ 3. & 2. & 5. & 4. \end{pmatrix}$$

where the variable indices are indicated by the integers before each matrix (columns are identified symmetrically to rows).

In assembled form this matrix is as follows:

$$\begin{pmatrix} 6. & 1. & & 2. & 3. & \\ 1. & 7. & 5. & 3. & 10. & 2. \\ & 5. & 4. & & 3. & 2. \\ 2. & 3. & & 6. & 4. & \\ 3. & 10. & 3. & -2. & 13. & 3. \\ & 2. & 2. & & 5. & 11. \end{pmatrix}$$

4.1 Elemental format

After reading a matrix in elemental format using `HSL_MC56`, the following components of the derived type `zd13_type` are set:

`matrix%n` is the largest index used to index a variable in the element matrices.

`matrix%nelt` is the number of elements.

`matrix%starts(:)` is allocated to have size `matrix%nelt+1`, and `matrix%starts(elt)` has been set to the position within `matrix%vars(:)` of the first variable in element `elt`. `matrix%starts(matrix%nelt+1)` has been set to one more than the position of the last variable of the final element.

`matrix%vars(:)` is allocated to have size equal to the number of variables. For each element `elt`, the array slice `matrix%vars(matrix%starts(elt):matrix%starts(elt+1)-1)` holds the associated variable list.

`matrix%values(:)` is allocated to have size equal to the number of entries if values are to be returned (otherwise it is unallocated). Elements are stored consecutively, and element `elt` has a number of values equal to either `n**2` in the unsymmetric (or full symmetric) case, or `n(n+1)` in the symmetric case, where `n=(matrix%starts(elt+1)-matrix%starts(elt))`. For each element the variables are stored as either a dense matrix in the unsymmetric case, or a packed matrix in the symmetric case, both by columns.

For example, reading the above matrix is equivalent to the following:

```
matrix%n = 6
matrix%nelt = 4
allocate(matrix%starts(5), matrix%vars(12), matrix%values(40))
matrix%starts(:) = (/ 1, 3, 5, 9, 13 /)
matrix%vars(:) = (/ 4, 5,  5, 6,  4, 5, 1, 2,  5, 6, 2, 3 /)
matrix%values(1:4) = (/ 2., -1.,  1., 7. /)
matrix%values(5:8) = (/ 3., 4.,  2., 8. /)
matrix%values(9:24) = (/ 4., ..., 5. /)
matrix%values(25:40) = (/ 2., ..., 4. /)
```

4.2 Coordinate format

Coordinate format stores an assembled matrix as a series of non-zero entries. A list of triples (i, j, v) is stored that indicates $A_{ij} = v$. If an entry corresponding to A_{ij} does not occur in the list, then $A_{ij} = 0$. After reading a matrix in assembled format using `HSL_MC56` with `control%format=3`, the following components of the derived type `zd11_type` are set:

`matrix%m` is the number of rows in the matrix

`matrix%n` is the number of columns in the matrix

`matrix%ne` is the number of entries in the matrix

`matrix%row(:)`, `matrix%col(:)` and `matrix%val(:)` are allocated to have size `matrix%ne`. The triple (`matrix%row(i)`, `matrix%col(i)`, `matrix%val(i)`) represents the row, column and value of the `i`-th entry of the matrix. They need not be in any particular order. If only the pattern is to be returned, then `matrix%val(:)` will be unallocated.

For example, reading the above matrix is equivalent to the following:

```
matrix%n = 6
matrix%m = 6
matrix%ne = 28
allocate(matrix%rows(28), matrix%cols(28), matrix%val(28))
matrix%row(:) = (/ 1,  2,  4,  5,  1,  2,  3,  4,   5,  6, ..., 6 /)
matrix%col(:) = (/ 1,  1,  1,  1,  2,  2,  2,  2,   2,  2, ..., 6 /)
matrix%val(:) = (/ 6., 1., 2., 3., 1., 7., 5., 3., 10., 2., ..., 11. /)
```

4.3 Compressed Sparse Column (CSC)

Compressed sparse column (CSC) format stores columns of the matrix consecutively as sparse vectors. That is to say that each column of the matrix is stored as a list of row indices and associated values. If row i is not present in the list for column j, then $A_{ij} = 0$. After reading a matrix in assembled format using `HSL_MC56` with `control%format=1`, the following components of the derived type `zd11_type` are set:

`matrix%m` is the number of rows in the matrix

`matrix%n` is the number of columns in the matrix

`matrix%ne` is the number of entries in the matrix

`matrix%ptr(:)` is allocated to have size `matrix%n+1`, and `matrix%ptr(col)` is set to the position within `matrix%row(:)` of the first variable in column `col`. `matrix%starts(matrix%n+1)` is set to one more than the position of the last variable of the final column.

`matrix%row(:)` and `matrix%val(:)` are allocated to have size equal to the number of entries. For each column `col`, the array sections `matrix%row(matrix%ptr(col):matrix%ptr(col+1)-1)` and `matrix%val(matrix%ptr(col):matrix%ptr(col+1)-1)` hold the associated row indices and values respectively. If only the pattern is to be returned, then `matrix%val(:)` will be unallocated.

For example, reading the above matrix is equivalent to the following:

```
matrix%n = 6
matrix%m = 6
matrix%ne = 28
allocate(matrix%ptr(7), matrix%row(28), matrix%val(28))
matrix%ptr(:) = (/ 1, 5, 11, 15, 19, 25, 29 /)
matrix%row(:) = (/ 1,  2,  4,  5,   1,  2,  3,  4,   5,  6, ..., 6 /)
matrix%val(:) = (/ 6., 1., 2., 3.,  1., 7., 5., 3., 10., 2., ..., 11. /)
```

4.4 Compressed Sparse Row (CSR)

Compressed sparse row (CSR) is similar to CSC, except the matrix is stored by rows rather than columns. After reading a matrix in assembled format using `HSL_MC56` with `control%format=2`, the following components of the derived type `zd11_type` are set:

`matrix%m` is the number of rows in the matrix

`matrix%n` is the number of columns in the matrix

`matrix%ne` is the number of entries in the matrix

`matrix%ptr(:)` is allocated to have size `matrix%n+1`, and `matrix%ptr(col)` is set to the position within `matrix%col(:)` of the first variable in column `row`. `matrix%starts(matrix%n+1)` is set to one more than the position of the last variable of the final row.

`matrix%col(:)` and `matrix%val(:)` are allocated to have size equal to the number of entries. For each row `row`, the array slices `matrix%col(matrix%ptr(row):matrix%ptr(row+1)-1)` and `matrix%val(matrix%ptr(row):matrix%ptr(row+1)-1)` hold the associated column indices and values respectively. If only the pattern is to be returned, then `matrix%val(:)` will be unallocated.

For example, reading the above matrix is equivalent to the following:

```
   matrix%n = 6
   matrix%m = 6
   matrix%ne = 28
   allocate(matrix%ptr(7), matrix%col(28), matrix%val(28))
   matrix%ptr(:) = (/ 1, 5, 11, 15, 19, 25, 29 /)
   matrix%col(:) = (/ 1,  2,  4,  5,   1,  2,  3,  4,   5,  6, ..., 6 /)
   matrix%val(:) = (/ 6., 1., 2., 3.,  1., 7., 5., 3., 10., 2., ..., 11. /)
```

5 EXAMPLE OF USE

The following program demonstrates the use of HSL_MC56 to read an unsymmetric matrix:

```
program hsl_mc56ds
   use hsl_mc56_double
   use hsl_zd11_double
   use hsl_zd13_double
   implicit none

   type(zd11_type) :: assembled_matrix ! Type for reading assembled matrices
   type(zd13_type) :: elemental_matrix ! Type for reading elemental matrices
   type(mc56_control) :: control       ! Type for hsl_mc56 control parameters
   integer :: info                     ! Return flag for hsl_mc56 subroutines
   character(len=3) :: type_code       ! String used to hold file data type
   character(len=72) :: title          ! String used to hold matrix title
   integer :: i, j, k                  ! Temporary variables
   integer :: vptr                     ! Pointer into elemental_matrix%values

   ! Determine if this is an assembled or elemental matrix
   call mc56_peek("hsl_mc56ds.data", info, type_code=type_code)

   ! Read into correct data type
   if(type_code(3:3).eq."a") then
      ! Assembled matrix
      call mc56_read("hsl_mc56ds.data", assembled_matrix, control, info, &
         title=title)
      write(*, "(a)") title
      write(*, "(a,3(i4,a))") "Assembled matrix of size ", assembled_matrix%m, &
         "x", assembled_matrix%n, " (ne = ", assembled_matrix%ne, ")"
      do i = 1, assembled_matrix%n
         j = assembled_matrix%ptr(i)
         k = assembled_matrix%ptr(i+1)-1
         write(*,"(a,i4,a)",advance="no") "Col ", i, " row : "
         write(*,"(10i6)") assembled_matrix%row(j:k)
         write(*,"(8x,a)",advance="no") " val : "
         write(*,"(10f6.2)") assembled_matrix%val(j:k)
      end do
   else ! type_code(3:3) = "e"
      ! Elemental matrix
      call mc56_read("hsl_mc56ds.data", elemental_matrix, control, info, &
         title=title)
      write(*, "(a)") title
      write(*, "(a,3(i4,a))") "Elemental matrix of size ", elemental_matrix%n, &
         "x", elemental_matrix%n, ", ", elemental_matrix%nelt, " elements"
      vptr = 1
      do i = 1, elemental_matrix%nelt
         j = elemental_matrix%starts(i)
         k = elemental_matrix%starts(i+1)-1
         write(*,"(a,i4,a)",advance="no") "Element ", i, " vars : "
         write(*,"(10i6)") elemental_matrix%vars(j:k)
         write(*,"(8x,a)",advance="no") " values : "
         write(*,"(10f6.2)") elemental_matrix%values(vptr:vptr+(k-j+1)**2-1)
         vptr = vptr + (k-j+1)**2
      end do
   endif
end program hsl_mc56ds
```

If we consider the example used in the previous section, and place the following data in the file `hsl_mc56ds.data`:

```
HSL_MC56DS assembled test example                                       MC56DS
             6             1             1             4
rua                        6             6            28             0
          (20i3)          (30i2)          (8es9.1)
  1  5 11 15 19 25 29
 1 2 4 5 1 2 3 4 5 6 2 3 5 6 1 2 4 5 1 2 3 4 5 6 2 3 5 6
  6.0E+00  1.0E+00  2.0E+00  3.0E+00  1.0E+00  7.0E+00  5.0E+00  3.0E+00
  1.0E+01  2.0E+00  5.0E+00  4.0E+00  3.0E+00  2.0E+00  2.0E+00  3.0E+00
  6.0E+00 -2.0E+00  3.0E+00  1.0E+01  3.0E+00  4.0E+00  1.3E+01  5.0E+00
  2.0E+00  2.0E+00  3.0E+00  1.1E+01
```

then running the program will produce the following output:

```
HSL_MC56DS assembled test example
Assembled matrix of size    6x   6 (ne =   28)
Col    1 row :      1     2     4     5
         val :   6.00  1.00  2.00  3.00
Col    2 row :      1     2     3     4     5     6
         val :   1.00  7.00  5.00  3.00 10.00  2.00
Col    3 row :      2     3     5     6
         val :   5.00  4.00  3.00  2.00
Col    4 row :      1     2     4     5
         val :   2.00  3.00  6.00 -2.00
Col    5 row :      1     2     3     4     5     6
         val :   3.00 10.00  3.00  4.00 13.00  5.00
Col    6 row :      2     3     5     6
         val :   2.00  2.00  3.00 11.00
```

However, if we instead place the following data in the file `hsl_mc56ds.data`:

```
HSL_MC56DS elemental test matrix                                        MC56DS1
             7             1             1             5
```

```
rue                        6             4            12            40
(26I3)          (40I2)              (8es9.1)
  1  3  5  9 13
 4 5 5 6 4 5 1 2 5 6 2 3
  2.0E+00 -1.0E+00  1.0E+00  7.0E+00  3.0E+00  4.0E+00  2.0E+00  8.0E+00
  4.0E+00 -1.0E+00  2.0E+00  3.0E+00  3.0E+00  1.0E+00  3.0E+00  2.0E+00
  2.0E+00  3.0E+00  6.0E+00  1.0E+00  3.0E+00  2.0E+00  1.0E+00  5.0E+00
  2.0E+00  1.0E+00  8.0E+00  3.0E+00  1.0E+00  3.0E+00  2.0E+00  2.0E+00
  8.0E+00  2.0E+00  2.0E+00  5.0E+00  3.0E+00  2.0E+00  5.0E+00  4.0E+00
```

then we get the following output:

```
HSL_MC56DS elemental test matrix
Elemental matrix of size    6x   6,    4 elements
Element    1 vars :      4     5
         values :   2.00 -1.00  1.00  7.00
Element    2 vars :      5     6
         values :   3.00  4.00  2.00  8.00
Element    3 vars :      4     5     1     2
         values :   4.00 -1.00  2.00  3.00  3.00  1.00  3.00  2.00  2.00  3.00
  6.00  1.00  3.00  2.00  1.00  5.00
Element    4 vars :      5     6     2     3
         values :   2.00  1.00  8.00  3.00  1.00  3.00  2.00  2.00  8.00  2.00
  2.00  5.00  3.00  2.00  5.00  4.00
```