HSL_MA77

1 SUMMARY

`HSL_MA77` solves one or more sets of sparse symmetric equations $\mathbf{AX} = \mathbf{B}$ using an out-of-core multifrontal method. The symmetric matrix $\mathbf{A}$ may be either positive definite or indefinite. It may be input by the user in either of the following ways:

(i) by square symmetric elements, such as in a finite-element calculation, or

(ii) by rows.

In both cases, the coefficient matrix is of order n and is of the form

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)}.$$

In (i), the summation is over elements and $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables in the kth element. In (ii), the summation is over rows and $\mathbf{A}^{(k)}$ is nonzero only in row k. In both cases, for each k, the user must supply a list specifying which columns of $\mathbf{A}$ are associated with $\mathbf{A}^{(k)}$, and an array containing $\mathbf{A}^{(k)}$ in packed form. This is defined more precisely in Sections 2.4.5 and 2.4.7 (arguments `list` and `reals`). It is permissible for some of the rows and corresponding columns to be empty, that is, to appear in none of the matrices $\mathbf{A}^{(k)}$; such rows and columns are ignored in determining whether the matrix is positive definite or singular.

The multifrontal method is a variant of sparse Gaussian elimination. In the positive-definite case, it involves the Cholesky factorization

$$\mathbf{A} = (\mathbf{PL})(\mathbf{PL})^T$$

where $\mathbf{P}$ is a permutation matrix and $\mathbf{L}$ is lower triangular. In the indefinite case, it involves the factorization

$$\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^T$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is unit lower triangular, and $\mathbf{D}$ is block diagonal with blocks of size 1×1 and 2×2. The factorization is performed by the subroutine `MA77_factor` and is controlled by an elimination tree that is constructed by the subroutine `MA77_analyse`, which needs the lists of variables in elements or rows and an elimination sequence. Once a matrix has been factorized, any number of calls to the subroutine `MA77_solve` may be made for different right-hand sides $\mathbf{B}$. An option exists for computing the residuals. For large problems, the matrix data and the computed factors are held in direct-access files.

The efficiency of `HSL_MA77` is dependent on the elimination order that the user supplies. The HSL routine `HSL_MC68` may be used to obtain a suitable ordering.

All the data for a problem are held in a structure `keep` and the files that it accesses. It is therefore possible to have more than one problem active at the same time. For each problem, it is permitted to change the real data, in which case a new call of `MA77_factor` is needed. Any change to the integer data, however, must be treated as creating a new problem to be input afresh.

For a very large problem, several direct-access files are used. The actual input/output is performed through the package `HSL_OF01`. This automatically shares the available memory in units called *pages*, whose size and number are under the user's control (see Section 2.4.19). If a file become full, `HSL_OF01` opens secondary files and treats the primary file and all its secondaries as a single superfile. To allow the secondary files to reside on different devices, the user may supply an array of path names; the full name of a file is the concatenation of a path name with the file name.

If the problem is not very large, the superfiles may be replaced by arrays in memory and the code is run in core. Storage is measured in Fortran storage units, with one unit for default reals and integers, and two units for double precision reals and long integers.

At the heart of the subroutines `MA77_factor` and `MA77_solve` there are calls to the packages `HSL_MA54` and `HSL_MA64` for the efficient partial factorization and partial solution of full sets of symmetric positive definite and symmetric indefinite equations, respectively. These blocks the matrix to reduce caching overheads.

An option exists to scale the matrix. In this case, the factorization of the scaled matrix $\overline{\mathbf{A}} = \mathbf{SAS}$ is computed, where **S** is a diagonal scaling matrix.

ATTRIBUTES — Version: 6.1.0 (24 April 2015). **Interfaces:** Fortran, C. **Types:** Real (single, double). **Uses:** `KB07`, `HSL_KB22`, `HSL_OF01`, `HSL_MA54`, `HSL_MA64` and BLAS routines `_axpy`, `_copy`, `_gemv`, `_nrm2`, `_tpmv`. **Original date:** September 2006; Version 5.0.0. August 2009; Version 6.0.0. March 2013. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 2003 subset (F95 + TR15581). **Parallelism:** May use OpenMP through `HSL_MA54` and `HSL_MA64`. **Remark:** The development of `HSL_MA77` was supported by the two EPSRC grants GR/S42170 and EP/E053351/1.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a `USE` statement

Single precision version

```
        USE HSL_MA77_single
```

Double precision version

```
        USE HSL_MA77_double
```

If it is required to use more than one module at the same time, the derived types (Section 2.3) must be renamed in one of the `USE` statements.

The following procedures are available to the user:

- `MA77_open` must be called once for each problem to initialize the data structures and open the superfiles.
- `MA77_input_vars` must be called once for each element or row to specify which variables are associated with it.
- `MA77_analyse` must be called after all calls to `MA77_input_vars` are complete. The user must supply an elimination order that is used to construct the data structures needed for the factorization.
- `MA77_input_reals` must be called for each element or row to specify the entries of $\mathbf{A}^{(k)}$. For large problems, the data may be provided in more than one adjacent call. The call (or calls) to `MA77_input_reals` for a given element or row may be made at any time after the corresponding call to `MA77_input_vars`. All the reals must be input before `MA77_factor` or `MA77_factor_solve` is called. If the user enters data for an element or row that has previously been entered, the original data are discarded. If this is done after a call to `MA77_factor` or `MA77_factor_solve`, a new call to `MA77_factor` or `MA77_factor_solve` will be needed.
- `MA77_scale` may be called after all the reals of **A** have been input and after the call to `MA77_analyse`. If called, a scaling of the matrix is computed.
- `MA77_factor` may be called after all the reals of **A** have been input and after the call to `MA77_analyse`. The matrix **A** is factorized using the information from the call to `MA77_analyse`. Multiple calls to `MA77_factor` may follow a call to `MA77_analyse`.
- `MA77_factor_solve` may be called in place of `MA77_factor` to factorize **A** and, at the same time, solve the system $\mathbf{AX} = \mathbf{B}$. Multiple calls to `MA77_factor_solve` may follow a call to `MA77_analyse`.

- `MA77_solve` uses the computed factors generated by `MA77_factor` or `MA77_factor_solve` to solve systems $\mathbf{AX} = \mathbf{B}$. Multiple calls to `MA77_solve` may follow a call to `MA77_factor` or `MA77_factor_solve`. An option is available to perform a partial solution.
- `MA77_resid` may be called after a call to `MA77_factor_solve` or after a call to `MA77_solve`. It computes the residual matrix $\mathbf{B} - \mathbf{AX}$.
- `MA77_finalise` should be called after all other calls are complete for a problem (including after an error return that does not allow the computation to continue). By default, it deallocates the components of the derived data types and discards the files associated with the problem. An option exists to close but keep the files and to write to another file the components of the derived data types that are needed if the user later wishes to restart the computation after a successful factorization.
- `MA77_restart` may be called after a call to `MA77_finalise` that filed the problem data. It restarts the computation and allows the user to solve for further right-hand sides or factorize another matrix with the same structure.
- `MA77_enquire_posdef` may be called in the positive definite case to obtain the pivots used.
- `MA77_enquire_indef` may be called in the indefinite case to obtain the pivot sequence used by the factorization and the entries of $\mathbf{D}^{-1}$.
- `MA77_alter` may be called in the indefinite case to alter the entries of $\mathbf{D}^{-1}$. Note that this means a factorization of $\mathbf{A}$ is no longer available.
- `MA77_solve_fredholm` is an alternative solve routine that may be called in the indefinite case when the matrix $\mathbf{A}$ is found to be singular. It computes the same solution $\mathbf{X}$ as `MA77_solve` but, if the j-th system is inconsistent, it also returns $\mathbf{Y}_j$ that satisfies $\mathbf{AY}_j = 0$ and $\mathbf{Y}_j^T\mathbf{B}_j \neq 0$.
- `MA77_lmultiply` may be called after a call to `MA77_factor` or `MA77_factor_solve` to perform the matrix-matrix product $\mathbf{PLX}$ or $(\mathbf{PL})^T\mathbf{X}$.

2.2 OpenMP

OpenMP is used by the `HSL_MA77` package to provide parallelism for shared memory environments.

If OpenMP is available then it should be enabled at compilation time by using the correct compiler flag (usually some variant of `-openmp`). The supplied replacement module `omp_lib` **must not** be compiled.

If OpenMP is *not* available then the user **must** compile with the supplied replacement module `omp_lib`.

2.3 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types `MA77_control`, `MA77_info`, and `MA77_keep`. The following pseudocode illustrates this.

```
use HSL_MA77_double
...
type (MA77_control) :: control
type (MA77_info) :: info
type (MA77_keep) :: keep
...
```

The components of `MA77_control` and `MA77_info` are explained in Sections 2.4.19 and 2.4.20. The components of `MA77_keep` are used to pass data between the subroutines of the package.

2.4 Argument lists and calling sequences

2.4.1 Optional arguments

We use square brackets `[ ]` to indicate `OPTIONAL` arguments. In each call, optional arguments follow the argument `info`. Since we reserve the right 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**.

2.4.2 Integer and real kinds

`INTEGER(short)` denotes default `INTEGER` and `INTEGER(long)` denotes `INTEGER(kind=selected_int_kind(18))`. `REAL` denotes default real in the single precision version and double precision real in the double precision version.

2.4.3 32-bit and 64-bit architectures

By default, it is assumed that the architecture is 32-bit. The parameter `control%bits` should be set to 64 if the user is running on a 64-bit architecture. On a 32-bit architecture, the maximum size of a rank-1 `REAL` array that can be allocated is taken to be huge(0_short)/4 in the single precision version and huge(0_short)/8 in the double precision version, where `huge` is the Fortran inquiry function. On a 64-bit architecture, it is taken to be huge(0_long)/4 and huge(0_long)/8, respectively.

2.4.4 The initialization subroutine

Data structures are set up and superfiles are opened by a call to `MA77_open`.

```
call MA77_open(n,filename,keep,control,info[,nelt,path])
```

`n` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that must be set to the matrix order. **Restriction:** `n`≥`0`.

`filename` is an array `INTENT(IN)` argument of size `4`, type `CHARACTER` and character length at most 400. `filename(1)` and `filename(2)` identify the integer and real superfiles that are used to hold the matrix and factor data; `filename(3)` identifies the superfile that is used for real workspace; `filename(4)` identifies the superfile that is used in the indefinite case as workspace during the numerical factorization. `filename(1:4)` must all be different. For each superfile named `filename(j)` that is opened by `HSL_OF01` and used by `HSL_MA77`, the name of the primary file is `filename(j)` or, if `path` is present, `path(i)//filename(j)` for an element `i` of `path`. Secondary files have names that are constructed by appending `1`, `2`, ... to this form, perhaps with a different element of `path`. Note that identifiers for **all** the superfiles must be provided even if the user wishes to solve only positive-definite problems and/or to work in-core (see Section 2.6). **Restriction:** `len(filename)`≤`400`.

`keep` is a scalar `INTENT(OUT)` argument of type `MA77_keep`. It is used to hold data about the problem being solved and must be passed unchanged to the other subroutines. `MA77_open` allocates its allocatable components.

`control` is a scalar `INTENT(IN)` argument of type `MA77_control`. Its components control the actions of the package, see Section 2.4.19.

`info` is a scalar `INTENT(OUT)` argument of type `MA77_info`. Its components provide information about the execution of the package, as explained in Section 2.4.20.

`nelt` is a optional scalar `INTENT(IN)` argument of type `INTEGER(short)`. If input is by elements, `nelt` must be set to be at least the largest integer used to index an element. If `nelt` is absent, input is by rows. **Restriction:** `nelt`≥`0`.

`path` is an optional assumed-shape array `INTENT(IN)` argument of type `CHARACTER` and character length at most 400. If `path` is absent, the behaviour is as if it were present with the value `(/''/)`. If present, the user must supply in `path` path names for the direct-access files; the path name must end with a `/`. By supplying more than one path name, the primary and secondary files can reside on different devices. The value `/''/` is permitted for an element of `path`. If `size(path) > 1`, there is a check for each new file to make sure that there is room for it, which may involve a significant overhead (see Section 4). **Restriction:** `len(path)`$\leq$`400`.

2.4.5 The input of integer data

A call of the following form must be made for each element or row:

```
call MA77_input_vars(index,nvar,list,keep,control,info)
```

`index` is a scalar `INTENT(IN)` argument of type `INTEGER(short)`. It must hold the index of the incoming element or row. Each element or row may be input only once (it is **not** possible to change the variable list for an element or row without first calling `MA77_finalise` to terminate the computation, recalling `MA77_open` and then recalling `MA77_input_vars` for each element or row). If `index` is out-of-range, the element or row is ignored.

`nvar` is a scalar `INTENT(IN)` argument of type `INTEGER(short)`. It must hold the number of variables in the incoming element or row. **Restriction:** `nvar`$\geq$`0`.

`list` is an array `INTENT(IN)` argument of size at least `nvar` of type `INTEGER(short)`. It must hold the indices of the variables in the incoming element or row. Duplicates are allowed. Out-of-range indices are ignored.

`keep` is a scalar `INTENT(INOUT)` argument of type `MA77_keep` that must be passed unchanged by the user.

`control` is a scalar `INTENT(IN)` argument of type `MA77_control` (see Section 2.4.19).

`info` is a scalar `INTENT(INOUT)` argument of type `MA77_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.4.20. It must be passed unchanged by the user.

2.4.6 To analyse the sparsity pattern and prepare for the factorization

After completion of the sequence of calls to `MA77_input_vars`, a call of the following form must be made:

```
call MA77_analyse(order,keep,control,info)
```

`order` is an array `INTENT(INOUT)` argument of size at least `n` of type `INTEGER(short)`. It must specify the elimination order. If `i` is used to index a variable, `abs(order(i))` must hold its position in the pivot sequence. If a 1×1 pivot i is required, the user must set `order(i)`$>$`0`. If a 2×2 pivot involving variables i and j is required, the user must set `order(i)`$<$`0`, `order(j)`$<$`0` and $|\texttt{order(j)}| = |\texttt{order(i)}| + 1$. If `i`, `1`$\leq$`i`$\leq$`n`, is not used to index a variable, `order(i)` may have any value and this is replaced by zero. On exit, `order` contains the elimination order that `MA77_factor` or `MA77_factor_solve` will be given; this order may give slightly more fill-in than the user-supplied order and, in the indefinite case, may be modified by `MA77_factor` or `MA77_factor_solve` to maintain numerical stability. Note that 2×2 pivots are only appropriate if the matrix **A** is **not** positive definite.

`keep`, `control`, `info`: see Section 2.4.5.

2.4.7 The input of real data

The subroutine `MA77_input_reals` must be called for each element or row to specify its reals. The corresponding integer data must have already been entered. In the element case, the data must be packed in the order given by the integer data into the lower-triangular part of a full symmetric matrix held by columns with no gaps between columns. In the row case, the reals for the whole row must be packed in the same order as the integer data into a full vector (**both upper and lower triangular entries must be supplied**). The two forms of entry are illustrated in Section 5. The reals for each element or row may be provided using a single call or, if the index lists that were passed to `MA77_input_vars` contained no duplicated or out-of-range indices, using a sequence of adjacent calls. Any previous real data for the element or row is discarded.

```
call MA77_input_reals(index,length,reals,keep,control,info)
```

`index` is a scalar `INTENT(IN)` argument of type `INTEGER(short)`. It must hold the index of the incoming element or row. If `index` is out-of-range, the element or row is ignored.

`length` is a scalar `INTENT(IN)` argument of type `INTEGER(short)`. It must hold the number of reals being input on this call. **Restriction:** `length`≥`0`.

`reals` is an array `INTENT(IN)` argument of size at least `length` of type `REAL`. It must hold the reals being input on this call.

`keep`, `control`, `info`: see Section 2.4.5.

2.4.8 To compute scaling factors

To compute a scaling of the matrix **A**, a call of the following form may be made after the calls to `MA77_input_reals` are complete and after the call to `MA77_analyse`:

```
call MA77_scale(scale,keep,control,info[,anorm])
```

`scale` is a rank-1 array of size at least `n` of `INTENT(OUT)` and type `REAL`. On exit, `scale(1:n)` contains the diagonal entries of the scaling matrix **S**.

`keep`, `control`, `info`: see Section 2.4.5.

`anorm` is an optional scalar `INTENT(OUT)` argument of type `REAL`. On exit, it holds $||\mathbf{A}||_\infty$ (regardless of the value of `control%infnorm`).

2.4.9 To factorize the matrix and optionally solve $\mathbf{AX} = \mathbf{B}$

To factorize the matrix, a call of the following form must be made after the calls to `MA77_input_reals` are complete:

```
call MA77_factor(pos_def,keep,control,info[,scale])
```

If the user wishes to solve at the same time as factorizing the matrix, he or she should instead make a call of the following form:

```
call MA77_factor_solve(pos_def,keep,control,info,nrhs,lx,x[,scale])
```

`pos_def` is a scalar `INTENT(IN)` argument of type `LOGICAL`. It should be set to `.true.` if **A** is known to be positive definite (ignoring any unused rows and columns) and to `.false.` otherwise. If `pos_def = .true.`, no numerical pivoting for stability is performed during the numerical factorization.

`keep`, `control`, `info`: see Section 2.4.5.

`nrhs` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that holds the number of right-hand sides. **Restriction:** `nrhs`≥`1`.

`lx` is a scalar argument of `INTENT(IN)` and type `INTEGER(short)` that must be set to the first extent of the array `x`. **Restriction:** `lx`≥`n`.

`x` is a rank-2 array with extents `lx` and `nrhs` of `INTENT(INOUT)` and type `REAL`. It must be set so that, for each non-empty row `i` (that is, for each `i` that is used to index a variable), `x(i,j)` holds the component of the right-hand side for variable `i` to the `j`th system. On exit, `x(i,j)` holds the solution for variable `i` to the `j`th system.

`scale` is an optional rank-1 array of size at least `n` of `INTENT(IN)` and type `REAL`. If present, `scale(1:n)` must contain the diagonal entries of the scaling matrix **S**.

2.4.10 To solve linear systems using the computed factors

After the call to `MA77_factor`, one or more calls of the following form may be made to solve $\mathbf{AX}=\mathbf{B}$. Partial solutions may be performed by appropriately setting the optional parameter `job`.

```
call MA77_solve(nrhs,lx,x,keep,control,info[,scale,job])
```

`nrhs` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that holds the number of right-hand sides. **Restriction:** `nrhs`≥`1`.

`lx` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that holds the first extent of `x`. **Restriction:** `lx`≥`n`.

`x` is a rank-2 array with extents `lx` and `nrhs` of `INTENT(INOUT)` and type `REAL`. It must be set so that, for each non-empty row `i` (that is, for each `i` that is used to index a variable), `x(i,j)` holds the component of the right-hand side for variable `i` to the `j`th system. On exit, `x(i,j)` holds the solution for variable `i` to the `j`th system.

`keep`, `control`, `info`: see Section 2.4.5.

`scale` is an optional rank-1 array of size at least `n` of `INTENT(IN)` and type `REAL`. If `scale` was present on the last call to `MA77_factor` (or `MA77_factor_solve`), it must also be present on each call to `MA77_solve` and `scale(1:n)` must be unchanged since the call to `MA77_factor` (or `MA77_factor_solve`).

`job` is an optional scalar `INTENT(IN)` argument of type `INTEGER(short)`. If absent, $\mathbf{AX}=\mathbf{B}$ is solved. In the positive-definite case, the Cholesky factorization that has been computed may be expressed in the form

$$\mathbf{SAS}=(\mathbf{PL})(\mathbf{PL})^T$$

where **P** is a permutation matrix and **L** is lower triangular. In the indefinite case, the factorization that has been computed may be expressed in the form

$$\mathbf{SAS}=(\mathbf{PL})\mathbf{D}(\mathbf{PL})^T$$

where **P** is a permutation matrix, **L** is unit lower triangular, and **D** is block diagonal with blocks of order 1 and 2. **S** is a diagonal scaling matrix and is equal to the identity unless `scale` was present on the last call to `MA77_factor` (or `MA77_factor_solve`). A partial solution may be computed by setting `job` to have one of the following values:

`1` for solving $\mathbf{PLX}=\mathbf{SB}$

`2` for solving $\mathbf{DX}=\mathbf{B}$ (indefinite case only)

`3` for solving $(\mathbf{PL})^T\mathbf{S}^{-1}\mathbf{X}=\mathbf{B}$

`4` for solving $\mathbf{D}(\mathbf{PL})^T\mathbf{S}^{-1}\mathbf{X}=\mathbf{B}$ (indefinite case only)

Restriction: `job = 1,2,3,4`.

2.4.11 To compute the residual matrix $\mathbf{B}-\mathbf{AX}$

Following a call to MA77_factor_solve or to MA77_solve (with job absent), the residual matrix $\mathbf{B}-\mathbf{AX}$ may be computed by making a call of the form:

```
call MA77_resid(nrhs,lx,x,lresid,resid,keep,control,info[,anorm_bnd])
```

nrhs is a scalar INTENT(IN) argument of type INTEGER(short) that holds the number of right-hand sides for which the residuals are required. **Restriction:** nrhs$\geq$1.

lx is a scalar INTENT(IN) argument of type INTEGER(short) that holds the first extent of x. **Restriction:** lx$\geq$n.

x is a rank-2 array with extents lxb and nrhs of INTENT(IN) and type REAL. It must be set to hold the matrix $\mathbf{X}$.

lresid is a scalar INTENT(IN) argument of type INTEGER(short) that holds the first extent of resid. **Restriction:** lresid$\geq$n.

resid is a rank-2 array with extents lresid and nrhs of INTENT(INOUT) and type REAL. It must be set to hold the matrix $\mathbf{B}$ and is overwritten by the matrix $\mathbf{B}-\mathbf{AX}$.

keep, control, info: see Section 2.4.5.

anorm_bnd is an optional scalar INTENT(OUT) argument of type REAL. On exit, it holds $||\mathbf{A}||_\infty$ in the row-entry case and $\sum_{k=1}^{m}||\mathbf{A}^{(\mathbf{k})}||_\infty$ in the element case, which is an upper bound for $||\mathbf{A}||_\infty$.

2.4.12 The finalisation subroutine

Once all other calls are complete for a problem, after an error return that does not allow the computation to continue, or to store a successful factorization for further use, a call of the following form should be made to deallocate allocatable components of the structure keep and to close the files opened by the package for the problem:

```
call MA77_finalise(keep,control,info[,restart_file])
```

keep is a scalar INTENT(INOUT) argument of type MA77_keep that must be passed unchanged. On exit, allocatable components will have been deallocated.

control is a scalar INTENT(IN) argument of type MA77_control. Only the components that control printing are accessed (see Section 2.4.19).

info is a scalar INTENT(INOUT) argument of type MA77_info. Its components provide information about the execution of the subroutine, as explained in Section 2.4.20.

restart_file is an optional scalar INTENT(IN) argument of type CHARACTER and character length at most 500. If it is not present, all files opened by the package are closed and deleted. If the user wishes to retain the matrix data and matrix factors so that further factorizations or solves can be performed later, restart_file must hold the name (including the full path name) of a sequential access file. Data that needs to be preserved to allow further solves or factorizations are written to this file. Files that have been used by HSL_MA77 and are identified by filename(1:2) are saved and those identified by filename(3:4) (see Section 2.4.4) are deleted. **Restriction:** len(restart_file)$\leq$500.

2.4.13 The restart subroutine

If the user wishes to perform further factorizations or to solve for further right-hand sides after a call to MA77_finalise, a call of the following form should be made:

```
call MA77_restart(restart_file,filename,keep,control,info[,path])
```

restart_file is an scalar INTENT(IN) argument of type CHARACTER and character length at most 500. It must hold the name of the sequential access file that contains the data that was written by a call to MA77_finalise. The file must not have been changed since then. On successful exit, this file will be closed and deleted. **Restriction:** len(restart_file)≤500.

filename is an array INTENT(IN) argument of size 4, type CHARACTER and character length at most 400. filename must hold identifiers for the superfiles. The data in the primary files that are identified by filename(1:2) and their secondaries, if any, must be unchanged since the call to MA77_finalise. There must be no files with names path(i)//filename(j) for j = 3, 4 and any value of i in the range 1, 2, ..., SIZE(path). **Restriction:** len(filename)≤400.

keep is a scalar INTENT(OUT) argument of type MA77_keep. On exit, its allocatable components that are required by MA77_solve or MA77_factor or MA77_factor_solve will have been allocated and the contents restored.

control is a scalar INTENT(IN) argument of type MA77_control. Only the components that control printing are accessed (see Section 2.4.19).

info is a scalar INTENT(INOUT) argument of type MA77_info. Its components provide information about the execution of the subroutine, as explained in Section 2.4.20.

path is an optional assumed-shape array INTENT(IN) argument of type CHARACTER and character length at most 400. path must hold the path names for the direct-access files that will be opened by HSL_OF01. **Restriction:** len(path)≤400.

2.4.14 To obtain information on the factorization (positive definite case)

After a successful call to MA77_factor or to MA77_factor_solve with posdef = .true., information on the pivots may be obtained using a call of the form

```
call MA77_enquire_posdef(d,keep,control,info)
```

d is an array INTENT(OUT) argument of size n and type REAL. The i-th pivot will be placed in d(i), i = 1,2,...,n.

keep, control, info: see Section 2.4.5.

2.4.15 To obtain information on the factorization (indefinite case)

After a successful call to MA77_factor or to MA77_factor_solve with posdef = .false., information on the pivot sequence and the matrix $\mathbf{D}^{-1}$ may be obtained using a call of the form

```
call MA77_enquire_indef(piv_order,d,keep,control,info)
```

piv_order is an array INTENT(OUT) argument of size n. If i is used to index a variable, its position in the pivot sequence will be placed in piv_order(i), with its sign negative if it is part of a 2×2 pivot. If i, 1≤i≤n, is not used to index a variable, piv_order(i) will be set to zero.

`d` is a rank-2 `INTENT(OUT)` argument of size `(2,n)` and type `REAL`. The diagonal entries of $\mathbf{D}^{-1}$ will be placed in `d(1,i), i = 1,2,...,n`, the off-diagonal entries of $\mathbf{D}^{-1}$ will be placed in `d(2,i), i = 1,2,...,n-1`, and `d(2,n)` will be set to zero.

`keep, control, info`: see Section 2.4.5.

2.4.16 To alter $\mathbf{D}^{-1}$

After a successful call to `MA77_factor` or to `MA77_factor_solve` with `posdef = .false.`, the matrix $\mathbf{D}^{-1}$ may be altered using a call of the form

```
call MA77_alter(d,keep,control,info)
```

`d` is a rank-2 `INTENT(IN)` argument of size `(2,n)` and type `REAL`. The diagonal entries of $\mathbf{D}^{-1}$ will be altered to `d(1,i), i = 1,2,...,n`, and the off-diagonal entries will be altered to `d(2,i), i = 1,2,...,n-1` (and $\mathbf{PL})\mathbf{D}(\mathbf{PL})^T$ will no longer be a factorization of $\mathbf{A}$).

`keep, control, info`: see Section 2.4.5.

2.4.17 To solve linear systems in the indefinite singular case

In the indefinite case, after the call to `MA77_factor`, one or more calls of the following form may be made. It computes the same solution $\mathbf{X}$ as `MA77_solve` but, if the j-th system is inconsistent, it also returns the Fredholm alternative solution $\mathbf{Y}_j$ that satisfies $\mathbf{AY}_j = 0$ and $\mathbf{Y}_j^T\mathbf{B}_j \neq 0$.

```
call MA77_solve_fredholm(nrhs,flag_out,lx,x,keep,control,info[,scale])
```

`nrhs, lx, keep, control, info, scale`: see Section 2.4.10.

`flag_out` is a rank-1 `INTENT(OUT)` argument of size `nrhs` and type `LOGICAL`. On exit, `flag_out(j)` is set to `.true.` if the $j-$th system is consistent and to `.false.` otherwise.

`x` is a rank-2 `INTENT(INOUT)` argument of package type with extents `lx` and `2*nrhs`. It must be set so that, for each non-empty row `i` (that is, for each `i` that is used to index a variable), `x(i,j)` holds the component of the right-hand side for variable `i` to the `j`-th system (`j=1,2,...,nrhs`). On exit, `x(:,1:nrhs)` holds the same solution as is returned by `MA77_solve` and, if `flag_out(j)=.false.`, `x(:,nrhs+j-1)` holds the Fredholm alternative solution for the `j`-th system.

2.4.18 To form a matrix product with PL or $(\mathbf{PL})^T$

After a call to `MA77_factor` or `MA77_factor_solve`, the product $\mathbf{Y} = \mathbf{PLX}$ or $\mathbf{Y} = (\mathbf{PL})^T\mathbf{X}$ (or $\mathbf{Y} = \mathbf{S}^{-1}\mathbf{PLX}$ or $\mathbf{Y} = (\mathbf{S}^{-1}\mathbf{PL})^T\mathbf{X}$ if `scale` was present on the call to `MA77_factor` or `MA77_factor_solve`) may be computed using a call of the following form

```
call MA77_lmultiply(trans,k,lx,x,ly,y,keep,control,info[,scale])
```

`trans` is a scalar `INTENT(IN)` argument of type `LOGICAL`. If set to `.true.`, $\mathbf{Y} = (\mathbf{PL})^T\mathbf{X}$ (or $\mathbf{Y} = (\mathbf{S}^{-1}\mathbf{PL})^T\mathbf{X}$) is computed; otherwise, $\mathbf{Y} = \mathbf{PLX}$ (or $\mathbf{Y} = \mathbf{S}^{-1}\mathbf{PLX}$) is computed.

`k` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that holds the number of columns of $\mathbf{X}$ and $\mathbf{Y}$. **Restriction:** `k`$\geq$`1`.

`x` is a rank-2 array with extents `lx` and `k` of `INTENT(INOUT)` and type `REAL`. It must be set to hold the matrix **X**. It is altered only if **`trans`** = **`.true.`** and `scale` is present (in this case, `x` holds $\mathbf{S}^{-1}\mathbf{X}$ on exit).

`y` is a rank-2 array with extents `ly` and `k` of `INTENT(OUT)` and type `REAL`. On exit, it holds the requested matrix product. Note that if variable `i` is not used to index a variable, `y(i,j)` is set to zero (`j` =`1,2,...,nrhs`).

`keep`, `control`, `info`, `scale`: see Section 2.4.10.

2.4.19 The derived data type for holding control parameters

The derived data type `MA77_control` is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are:

Printing controls

`print_level` is a scalar of type `INTEGER(short)` that is used to controls the level of printing. The different levels are:

- $<$ `0` No printing.
- = `0` Error and warning messages only.
- = `1` As 0, plus basic diagnostic printing.
- $>$ `1` As 1, plus some additional diagnostic printing.

The default is `print_level=0`.

`unit_diagnostics` is a scalar of type `INTEGER(short)` that holds the unit number for diagnostic printing. Printing is suppressed if `unit_diagnostics`<0. The default is `unit_diagnostics=6`.

`unit_error` is a scalar of type `INTEGER(short)` that holds the unit number for error messages. Printing of error messages is suppressed if `unit_error<0`. The default is `unit_error=6`.

`unit_warning` is a scalar of type `INTEGER(short)` that holds the unit number for warning messages. Printing of warning messages is suppressed if `unit_warning<0`. The default is `unit_warning=6`.

Controls used by `MA77_open`

`bits` is a scalar of type `INTEGER(short)` that indicates the machine architecture being used. It should be set to `32` on a 32-bit architecture and to `64` on a 64-bit architecture. The default value is `32`. Note that, if the maximum frontsize is found to exceed approximately 2^{14}, the computation must be performed on a 64-bit machine.

`buffer_lpage` is an array of size 2 of type `INTEGER(short)` that holds the number of scalars held in each page of the integer and real in-core buffers that are used by `HSL_OF01`. The default is `buffer_lpage(1:2)`= 2^{12}. **Restriction:** `1≤buffer_lpage(:)≤file_size`.

`buffer_npage` is an array of size 2 of type `INTEGER(short)` that holds the number of pages in the integer and reals in-core buffers that are used by `HSL_OF01`. The default is `buffer_npage(1:2)=1600`. **Restriction:** `buffer_npage(:)≥1`.

`file_size` is a scalar component of type `INTEGER(long)` that holds the target size of each file, measured in scalars of the package type. The actual size is `buffer_lpage*(file_size/buffer_lpage)`. The default is 2^{21}. N.B. This does not limit the size of a superfile which may consist of many files but there is a system limit on the number of open files and so, for very large problems, it may be necessary to use a larger value of `file_size` to prevent this limit from being reached.

maxstore is a scalar of type INTEGER(long) that holds the maximum amount of storage (measured in Fortran storage units) to be used if the user wants to use arrays in place of the superfiles. Further details are given in Section 2.6. The default is maxstore=0. **Restriction:** maxstore≥0. When running on 32-bit architecture (control%bits=32), the value of maxstore must not exceed huge(0_short)/4 in the single precision version and huge(0_short)/8 in the double precision version, where huge is the Fortran inquiry function; on a 64-bit architecture, maxstore must not exceed huge(0_long)/4 in the single precision version and huge(0_long)/8 in the double precision version.

storage is an array of size 3 and type INTEGER(short). If the user wants to use arrays in place of the superfiles filename(1:3), storage(1:3) should be set to the initial sizes for these arrays; if any of the sizes are zero, guesses based on the value of the component maxstore will be made. If an array is found to be not large enough, a superfile may be used instead. storage is not accessed if maxstore=0. If storage(i) < 0, a superfile identified by filename(i) is used (i = 1,2,3). Further details are given in Section 2.6. The default is storage(:)=0.

Controls used by MA77_analyse

nemin is a scalar of type INTEGER(short) that controls node amalgamation. Two neighbours in the elimination tree are merged if they both involve fewer than nemin eliminations. The default is nemin=8. The default is used if nemin<1.

Controls used by MA77_scale

maxit is a scalar of type INTEGER(short) that specifies the maximum number of iterations performed by the scaling algorithm. The default is maxit=1. The default is used if maxit<1.

infnorm is a scalar of type INTEGER(short) that controls the norm used by the scaling algorithm. If set to 0, the infinity norm is used; otherwise the one norm is used. The default is infnorm=0.

thresh is a scalar of type REAL and default value 0.5. The scaling algorithm terminates once the infinity (or one) norm of each row (and column) of the scaled matrix lies between 1±thresh.

Controls used by MA77_factor **with** pos_def **true**

nb54 is a scalar of type INTEGER(short) that holds the block size used within the kernel factorization code HSL_MA54. This is discussed further in Section 2.2.6 of the HSL_MA54 specification. The default is nb54=150. The default is used if nb54<1.

Controls used by MA77_factor **with** pos_def **false**

action is a scalar of type default LOGICAL. If the matrix is found to be singular (have rank less than the number of non-empty rows), the computation continues after issuing a warning if action has the value .true. or terminates (see error -11) if it has the value .false. The default is action=.true.

multiplier is a scalar of type REAL. To allow for delayed pivots, the arrays that hold the frontal matrix and its index list are allocated to accommodate a matrix of order $s \times max(1,\texttt{multiplier})$ if it is known that the front size will reach s. If, during the factorization, the arrays that hold the frontal matrix are found to be too small, they are reallocated to accommodate a matrix of order $s_c \times max(1,\texttt{multiplier})$, where s_c is the current front size. The default value is 1.1.

`nb64` is a scalar of type `INTEGER(short)` that holds the block size used within the kernel factorization code `HSL_MA64`. This is discussed further in Section 2.2.4 of the `HSL_MA64` specification. The default is `nb64=120`. The default is used if `nb64<1`. `nb64` and `nbi` must satisfy `mod(nb64,nbi)=0`. If this restriction is not satisfied, the block size is taken to be `max(nbi,nb64/nbi*nbi)`.

`nbi` is a scalar of type `INTEGER(short)` that holds the inner block size used within the kernel factorization code `HSL_MA64`. This is discussed further in Section 2.2.4 of the `HSL_MA64` specification. The default is `nbi=40`. The default is used if `nbi<1`.

`small` is a scalar of type `REAL`. Any pivot whose modulus is less than `small` is treated as zero. The default is `small` $= 1\times 10^{-20}$.

`static` is a scalar of type `REAL` that is used in the indefinite case only. It is used to control static pivoting within the kernel `HSL_MA64`. If `static>0.0` and if, at any stage of the computation, fewer than the expected number of pivots can be found with relative pivot tolerance greater than `umin`, diagonal entries are accepted as pivots. If a candidate diagonal entry has absolute value at least `static`, it is selected as a pivot; otherwise, the pivot is given the value that has the same sign but absolute value `static`. The default value is 0.0. **Restriction:** Either `static=0.0` or `static≥small`.

`storage_indef` is an array of size 2 and type `INTEGER(short)`. If the user wants to use arrays in place of the superfile `filename(4)`, `storage_indef(1:2)` should be set to the initial sizes for these arrays; if any of the sizes are zero, guesses based on the value of the component `maxstore` will be made. If an array is found to be not large enough, a superfile may be used instead. `storage_indef` is not accessed if `maxstore=0`. If `storage_indef(i)<0`, a superfile identified by `filename(3+i)` is used (`i=1,2`). Further details are given in Section 2.6. The default is `storage_indef(:)=0`.

`u` is a scalar of type `REAL` that holds the initial value of the relative pivot tolerance u used within the kernel `HSL_MA64`. The default is `u=0.01`. Values outside the range $[0,1.0]$ are treated as the default. Further details are given in the documentation for `HSL_MA64`.

`umin` is a scalar of type `REAL` that holds the minimum value of the relative pivot tolerance used within the kernel `HSL_MA64`. If, at any stage of the computation, fewer than the expected number of stable pivots have been found using the current tolerance u and the candidate pivot with greatest relative pivot tolerance has tolerance $v \geq$`umin`, this is accepted as a pivot and the tolerance u is set to v. The new value is used in subsequent calls to `HSL_MA64`. The default is `umin=1.0`. Values of `umin` greater than `u` are treated as `u` and values less than 0 are treated as 0.

Controls used by `MA77_solve_fredholm`

`consist_tol` is a scalar of type `REAL` and default value `epsilon(consist_tol)`. During the solve, a tolerance equal to `consist_tol` $* n * \|\mathbf{B}_i\|$ is used to determine whether the system of equations with the right-hand side $\mathbf{B}_i$ is consistent. It is only accessed if **A** is singular.

2.4.20 The derived data type for holding information

The derived data type `MA77_info` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `MA77_info` (in alphabetical order) are:

`detlog` is a scalar of type `REAL`. On exit from `MA77_factor` or `MA77_factor_solve`, it holds the logarithm of the absolute value of the determinant of **A** or zero if the determinant is zero.

`detsign` is a scalar of type `INTEGER(short)`. On exit from `MA77_factor` or `MA77_factor_solve`, it holds the sign of the determinant of **A** or zero if the determinant is zero.

flag is a scalar of type INTEGER(short) that gives the exit status of the algorithm (details in Section 2.5).

index is an array of length 4 and type INTEGER(short). On exit from MA77_open and MA77_restart, index(i) holds the HSL_OF01 index of the superfile identified by filename(i) (i=1,2,3,4). On exit from MA77_factor or MA77_factor_solve, index(i) has a negative sign if the superfile with index abs(index(i)) has not been used.

iostat is a scalar of type INTEGER(short) that holds the Fortran iostat parameter.

matrix_dup is a scalar of type INTEGER(short) that is set, on each exit from MA77_input_vars to the total number of duplicate entries that have been found.

matrix_outrange is a scalar of type INTEGER(short) that is set, on each exit from MA77_input_vars to the total number of out-of-range entries that have been found.

matrix_rank is scalar of type INTEGER(short). On exit from MA77_factor or MA77_factor_solve, it holds the computed rank of the factorized matrix.

maxdepth is a scalar of type INTEGER(short). On exit from MA77_analyse, it holds the maximum depth of the assembly tree.

maxfront is a scalar of type INTEGER(short). On exit from MA77_analyse, it holds the maximum front size in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from MA77_factor or MA77_factor_solve, it holds the maximum front size.

minstore is a scalar of type INTEGER(long). On exit from MA77_factor or MA77_factor_solve, it holds the amount of storage (measured in Fortran storage units) used in the superfiles (or in the arrays that replaced the superfiles). This is the least value for control%maxstore that would have permitted the computation to be performed in memory.

ndelay is scalar of type INTEGER(short). On exit from MA77_factor or MA77_factor_solve, it holds the number of eliminations that were delayed, that is, the total number of fully-summed variables that were passed to the father node because of stability considerations. If a variable is passed further up the tree, it will be counted again.

nfactor is scalar of type INTEGER(long). On exit from MA77_analyse, it holds the number of entries that will be in the factor **L** in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from MA77_factor or MA77_factor_solve, in the positive definite case, it holds the actual number of entries in the factor **L**. Note that, in the indefinite case, $2n$ entries of $\mathbf{D}^{-1}$ are also held.

nflops is scalar of type INTEGER(long). On exit from MA77_analyse, it holds the number of floating-point operations that will be needed to perform the factorization in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from MA77_factor or MA77_factor_solve, it holds the number of floating-point operations performed.

nio_read is an array of size 2 of type INTEGER(long). On exit from a call to MA77_analyse, MA77_factor, MA77_factor_solve, and MA77_solve, nio_read(1:2) holds the number of integer and real records actually read from disk by HSL_OF01 during the subroutine call.

nio_write is an array of size 2 of type INTEGER(long). On exit from a call to MA77_analyse, MA77_factor, MA77_factor_solve, and MA77_solve, nio_write(1:2) holds the number of integer and real records actually written to disk by HSL_OF01 during the subroutine call.

niter is a scalar of type INTEGER(short). On exit for MA77_scale, it holds the number of iterations of the scaling algorithm that were performed.

nsup is a scalar of type INTEGER(short). On exit from the final call to MA77_input_vars, it holds the number of supervariables in the problem (see Section 4).

ntwo is scalar of type INTEGER(short). On exit from MA77_analyse, it holds the number of 2×2 pivots in the pivot sequence. On exit from MA77_factor and MA77_factor_solve, it holds actual number of 2×2 used by the factorization, that is, the number of 2×2 blocks in **D**.

num_file is an array of size 4 of type INTEGER(short). On exit from a call to MA77_finalise, num_file(1:4) holds the number of secondary files used by each of the superfiles.

num_neg is a scalar of type INTEGER(short). On exit from MA77_factor or MA77_factor_solve, it holds the number of negative eigenvalues of the matrix **D**.

num_nothresh is a scalar of type INTEGER. On successful exit from MA77_factor or MA77_factor_solve, it holds the number diagonal entries of **D** that were chosen as 1×1 pivots without satisfying the relative pivot threshold criteria with relative pivot tolerance control%umin. It will have the value zero if control%static=0.0.

num_perturbed is a scalar of type INTEGER. On successful exit from MA77_factor or MA77_factor_solve, it holds the number of pivots that were perturbed to control%static or -control%static. It will have the value zero if control%static=0.0.

nwd_read is an array of size 2 of type INTEGER(long). On exit from a call to MA77_analyse, MA77_factor, MA77_factor_solve, and MA77_solve, nwd_read(1:2) holds the number of integer and real scalars read by HSL_OF01 during the subroutine call.

nwd_write is an array of size 2 of type INTEGER(long). On exit from a call to MA77_analyse, MA77_factor, MA77_factor_solve, and MA77_solve, nwd_write(1:2) holds the number of integer and real scalars written by HSL_OF01 during the subroutine call.

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

storage is an array of length 4 and type INTEGER(long). On exit from MA77_factor, MA77_factor_solve, it holds the maximum numbers of integers and reals that were stored in the superfiles filename(1:4) (or in the arrays that replaced the superfiles). In particular, storage(3) is the maximum multifrontal stack size.

tree_nodes is a scalar of type INTEGER(short). On exit from MA77_analyse, it holds the number of non-leaf nodes in the assembly tree (including any that are discarded but not reused).

unit_restart is a scalar of type INTEGER(short). On exit from MA77_finalise and MA77_restart it holds the unit number of the sequential access file with name restart_file.

unused is a scalar of type INTEGER(short). On exit from MA77_analyse, it holds the number of indices in the range 1 to n that were not used for variables.

u is a scalar of type REAL. On successful exit from MA77_factor or MA77_factor_solve, if num_perturbed=0, u holds the final value of the relative pivot tolerance *u* and is set to zero otherwise.

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%unit_error. If the error is such that another call of the same subroutine may be made immediately after the error has been corrected, we label the error as 'Immediate return'. Possible negative values are:

−1 Allocation error. The `stat` parameter is returned in `info%stat`. Note that if this error is returned when the user is attempting to use arrays instead of superfiles, it may be possible to avoid this error by using one or superfiles. If superfiles are being used (`control%maxstore = 0`), reducing `control%buffer_npage(:)` and/or `control%buffer_lpage(:)` may avoid this error.

−3 An error has been made in the sequence of calls.

−4 Returned by `MA77_open` if `n<0`.

−5 Error in Fortran `INQUIRE` statement. The `iostat` parameter is returned in `info%iostat`.

−6 Error in Fortran `READ`. The `iostat` parameter is returned in `info%iostat`.

−7 Error in Fortran `OPEN` statement. The `iostat` parameter is returned in `info%iostat`.

−8 Deallocation error. The `stat` parameter is returned in `info%stat`.

−9 Returned by `MA77_input_vars` if a call has already been made for the current element or row. Immediate return.

−10 Returned by `MA77_input_reals` if `MA77_input_vars` has not been called for the current element or row. Immediate return.

−11 Returned by `MA77_factor` and `MA77_factor_solve` if `pos_def = .true.` and the matrix is found to be not positive definite. This error is also returned if `pos_def = .false.` and `control%action = .false.` and the matrix is found to be singular. In both cases, empty rows and columns are ignored.

−12 Returned by `MA77_open` if a file of the given name already exists. This error is also returned by `MA77_finalise` if a file of name `restart_file` already exists and by `MA77_restart` if a file identified by `filename(3:4)` already exists.

−13 Returned by `MA77_open` or `MA77_restart` if `len(filename)>400`. This error is also returned by `MA77_finalise` if `len(restart_file)>500`.

−14 Returned by `MA77_input_reals` if the data for the previous element or row is incomplete. Immediate return.

−15 Error in Fortran `WRITE`. This can happen if there is insufficient space for one of the files. The `iostat` parameter is returned in `info%iostat`.

−16 Returned by `MA77_open` or `MA77_restart` if either `len(path)>400`. This error is also returned if a Fortran `OPEN` statement was not successful for any of the elements of `path` (either there is no room or a system limit on the number of open files has been reached). The `iostat` parameter is returned in `info%iostat`.

−17 Returned by `MA77_input_reals` if there are duplicated or out-of-range entries in one or more of the element or row variable lists and user has not entered all the reals for the current element or row in a single call to `MA77_input_reals`.

−18 Returned by `MA77_open` if `nelt<0`.

−19 Returned by `MA77_input_reals` if `length<0`. Immediate return.

−20 Returned by `MA77_solve` if `job` is out of range.

−21 Returned by `MA77_analyse` if an error is found in the user-supplied elimination order (held in `order`). Immediate return.

−22 Returned by `MA77_factor`, `MA77_factor_solve` and `MA77_scale` if for one or more of the elements or rows, `MA77_input_vars` was called but no corresponding call was made to `MA77_input_reals`.

−23 Returned by `MA77_open` if `control%buffer_lpage(:)<1` or `control%buffer_lpage(:)>control%file_size`. Immediate return.

−24 Returned by `MA77_factor`, `MA77_factor_solve`, `MA77_solve`, `MA77_solve_fredholm`, `MA77_resid`, and `MA77_lmultiply` if there is an error in the size of array `x` (that is, `lx<n` or `nrhs<1`). Also returned by `MA77_lmultiply` if there is an error in the size of array `y`.

−25 Returned by `MA77_resid` if there is an error in the size of array `resid`.

−26 Returned by `MA77_open` if `control%buffer_npage(:)<1`. Immediate return.

−27 Returned by `MA77_open` if `control%maxstore` is out of range. Immediate return.

−28 Returned by `MA77_restart` if a file with name `restart_file` does not exist. It is also returned if the expected files that are identified by `filename(1:2)` do not exist.

−29 Returned by `MA77_factor` and `MA77_factor_solve` if `pos_def=.true.` but the user supplied 2×2 pivots on the call to `MA77_analyse`.

−30 Returned by `MA77_factor`, `MA77_factor_solve` and `MA77_scale` if the front size is too large to successfully allocate the frontal matrix. To try and avoid this, the user should try running on a 64-bit architecture.

−31 Returned by `MA77_input_vars` if all the variable indices in an element/row are out-of-range.

−32 Returned by `MA77_input_reals` if more than the expected number of reals have been entered for the current element or row. Immediate return.

−33 Returned by `MA77_input_vars` if `nvar<0`. Immediate return.

−34 Returned by `MA77_enquire_indef` or `MA77_alter` if the call does not follow a successful call to `MA77_factor` or `MA77_factor_solve` with `pos_def=.false.`.

−35 Returned by `MA77_enquire_posdef` if the call does not follow a successful call to `MA77_factor` or `MA77_factor_solve` with `pos_def=.true.`.

−36 Returned by `MA77_enquire_posdef`, `MA77_enquire_indef`, or `MA77_alter` if there is an error in the size of the array `piv_order`.

−37 Returned by `MA77_enquire_posdef`, `MA77_enquire_indef`, or `MA77_alter` if there is an error in the size of the array `d`.

−38 Returned by `MA77_factor` and `MA77_factor_solve` if `control%static<control%small` and `control%static≠0.0`.

−39 Returned by `MA77_scale`, `MA77_factor`, `MA77_factor_solve`, and `MA77_solve` if the size of the array `scale` is too small. This error is also returned by `MA77_solve`, `MA77_solve_fredholm` and `MA77_lmultiply` if `scale` is absent when it was present on the call to `MA77_factor` (or `MA77_factor_solve`), or if `scale` is present when it was not present on the call to `MA77_factor` (or `MA77_factor_solve`).

−40 Returned by `MA77_factor` IEEE infinities found in the reduced matrix, probably caused by `control%small` or `control%u` having too small a value.

−41 Returned by `MA77_finalise` if there is an error in a Fortran `CLOSE` statement.

A positive value of `info%flag` on exit from `MA77_factor` or `MA77_factor_solve` is used to warn the user that the data may be faulty or that the subroutine cannot guarantee the solution obtained. Possible values are:

+1 Returned by `MA77_input_vars` if out-of-range variable indices have been found in the user-supplied array `list`. Any such entries are ignored and the computation continues. `info%matrix_outrange` is set to the number of such entries. Details of the first 10 are printed on unit `control%unit_warning`.

+2 Returned by `MA77_input_vars` if duplicated variable indices have been found in the user-supplied array `list`. Duplicates are recorded and the corresponding reals are summed by `MA77_input_reals`. `info%matrix_dup` is set to the number of such entries. Details of the first 10 are printed on unit `control%unit_warning`.

+3 Returned by `MA77_input_vars` if both out-of-range and duplicated variable indices have been found in the user-supplied array `list`.

+4 Returned by `MA77_factor` or `MA77_factor_solve` if `control%action = .true.` and the matrix is found to be singular.

2.6 In-core working

The user can request that arrays be used instead of superfiles by setting a value for `control%maxstore` and can specify initial sizes for the arrays in `control%storage(1:3)` and, if `MA77_factor` or `MA77_factor_solve` is called with `pos_def = .false.`, in `control%storage_indef(1:2)`. If `control%maxstore`>`0` and the user does not set `control%storage` and `control%storage_indef`, the code selects initial sizes for the arrays based on the value of `control%maxstore`. If an array is found to be too small, the code attempts to reallocate it with a larger size, provided the total for the five arrays (in Fortran storage units) does not exceed `control%maxstore`. Note the superfiles identified by `filename(1)` and `filename(4)` are for integers (one storage unit for each entry) and the rest are for reals (in the single precision version, one storage unit for each entry and in the double precision version, two storage units for each entry). If there is insufficient memory for an array, the contents of the array are written to a superfile and the in-core memory that was used by the array is freed (resulting in a combination of superfiles and in-core arrays being used). If the user sets `control%maxstore`>`0` and `control%storage(i)`<`0` for some `i = 1, 2,` or `3`, a superfile identified by `filename(i)` is used (allowing the user to choose to use, for example, an array for the integers and a superfile for the reals). Similarly, if `control%maxstore`>`0` and `control%storage_indef(i)` for `i = 1` or `2`, a superfile identified by `filename(3+i)` is used.

In some applications, a user may need to factorize a series of matrices of the same size and the same (or similar) sparsity pattern. The user may choose to run the first problem using the out-of-core facilities and may then use the information returned from that problem in `info%minstore` and `info%storage` to set the control parameters `control%maxstore`, `control%storage`, and `control%storage_indef` for subsequent runs.

If `MA77_finalise` is called with `restart_file` present, the matrix and factor integer and real data are written to superfiles identified by `filename(1:2)`. These files are read on a call to `MA77_restart`.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: `KB07`, `HSL_KB22`, `HSL_OF01`, `HSL_MA54`, `HSL_MA64`.

Input/output: Output is provided under the control of `control%print_level`. In the event of an error, diagnostic messages are printed. The output units for these messages are respectively controlled by `control%unit_err`, `control%unit_warning` and `control%unit_diagnostics` (see Section 2.4.19). I/O to direct-access files whose unit numbers are chosen by `HSL_OF01` and, if the restart facility is used, to a sequential access file whose unit number of chosen by `HSL_MA77`.

Restrictions: `n`$\geq$`0`; `nvar`$\geq$`0`; `len(path)`$\leq$`400`; `len(filename)`$\leq$`400`; `len(restart_file)`$\leq$`500`; `nrhs`$\geq$`1`; `lx`$\geq$`n`; `lresid`$\geq$`n`; `k`$\geq$`1`; `ly`$\geq$`n`; `1`$\leq$`buffer_lpage(:)`$\leq$`file_size buffer_npage(:)`$\geq$`1`; `control%maxstore`$\geq$`1`; `control%static = 0.0` or `control%static`$\geq$`control%small`.

Portability: Fortran 2003 subset (F95 + TR15581).

Changes from Version 1
Version 1 was only for positive definite systems; Version 2 can be used to solve positive definite or indefinite systems. Internally, the code has been revised so that recursive subroutines are no longer used.

Changes from Version 2
Version 3 allows the code to be used on a 64-bit machine.

Changes from Version 3
Version 4 includes an option to scale the matrix. The code has also been revised to use an updated version of `HSL_MA64`; this has included adding the control `control%nbi`.

Changes from Version 4
Following changes to `HSL_MA64`, the control component `umin` has been added and `control%u` is allowed in the range $[0, 1.0]$. In addition, `MA77_enquire_indef` and `MA77_alter` have been altered to work with $\mathbf{D}^{-1}$ (rather than with $\mathbf{D}$).

Changes from Version 5
The alternative solve routine `MA77_solve_fredholm` and the multiply routine `MA77_lmultiply` have been added.

4 METHOD

MA77_open

`MA77_open` must be called once for each problem. It initializes the data structures and calls `OF01_initialize` and `OF01_open` to open superfiles. The user must supply filenames even if he/she intends to work in-core. This is so that, if the in-core arrays are insufficient to successfully compute the factorization and the code is unable to successfully allocate in-core arrays that are large enough, the code is able to automatically switch to working (partly) out-of-core, without requiring the user the start the computation again.

The user may optionally supply pathnames for where the files are to be written on their system. If more than one pathname is supplied, the files may be held on different devices and this may allow the user to factorize larger problems than would be possible if all the files had to be held on a single device.

MA77_input_vars

`MA77_input_vars` must be called for each row or element to specify the `nvar` variables associated with it. The user's data is checked for errors and, if necessary, an error message is returned. In this case, the user should call `MA77_finalize`. Duplicates and out-of-range variable indices are allowed. A (possibly revised) list of variables with any out-of-range indices and duplicates removed is written to the main integer superfile. If the row or element contained any duplicated or out-or-range indices, a mapping array of length `nvar` that records the position of each variable in the row or element is also stored (a mapping to zero indicates the variable is out-of-range and will be ignored later).

Supervariables are constructed during the calls to `MA77_input_vars`.

Note that, having input the variable list for a particular row or element, the user may not input data for this row or element again without first calling `MA77_finalize` and then recalling `MA77_open` followed by `MA77_input_vars` for each row or element.

MA77_analyse

MA77_analyse must be called once after all the calls to MA77_input_vars are complete (it may be called before or after the calls to MA77_input_reals). The user must supply a pivot sequence in the array order. The HSL package HSL_MC68 may be used for this but note that HSL_MC68 currently requires the sparsity pattern of the assembled matrix to be input and so, in the element case, MC57 should be called first to assemble the sparsity pattern of **A**. The pivot order may contain 2x2 pivots. If a 2×2 pivot involving variables i and j is required, the user must set order(i)$<$0, order(j)$<$0 and $|\texttt{order(j)}| = |\texttt{order(i)}| + 1$.

The pivot order is used to construct the assembly tree. The list of variables for each node of the tree is stored as it is generated in the main integer superfile. Once the tree has been constructed, the children at each non-leaf node are ordered and the split point for the node (that is, the number of children that will be processed during the factorization before the assembly of the children into the frontal matrix is started) is computed.

Before returning to the user, order is reset so that |order(i)| holds the position at which variable i is eliminated. Finally, the variables at each non-leaf node of the tree are read back in, they are ordered into elimination order, and then written back to the main integer superfile. The position of the first free location in this superfile is held in a component keep.

MA77_input_reals

For each row or element, MA77_input_reals must be called, after the corresponding call to MA77_input_vars and before a call to MA77_factor or MA77_factor_solve. If the call to MA77_input_vars found duplicated or out-of-range indices, all the reals for that row or element must be input on a single call to MA77_input_reals, otherwise the input of the real data may be split over more than one call. Checks are made that the user has supplied all the real data for the previous row or element and that the number of reals does not exceed the expected number. If real data has already been supplied for the incoming row or element, it is overwritten by the new data (this allows the reals of a matrix to change without the using having to recall MA77_input_vars and MA77_analyse).

If duplicated or out-of-range indices were input on the call to MA77_input_vars, the compressed variable list and mapping array are read from the main integer superfile and used to sum entries corresponding to duplicated indices and to squeeze out the entries corresponding to out-of-range indices. The revised list of reals is stored in the main real superfile. The position of the first free location in this superfile is held in a component of keep.

MA77_scale

MA77_scale computes the scaling diagonal matrix **S** such that the infinity norm or one-norm of each row and column of $\overline{\mathbf{A}} = \mathbf{SAS}$ is approximately equal to 1. An iterative algorithm is used (control%maxit controls the maximum number of iterations); this is described in [2]. Each iteration involves reading the matrix once. This is expensive and so use of MA77_scale is only recommended if the user is unable to temporarily assemble the matrix and scale it using the HSL package MC77 before any routines from HSL_MA77 are called. Details of the scaling algorithm and how it is implemented within HSL_MA77 are given in [3]. MA77_scale includes an option to compute the infinity norm of the matrix **A**.

MA77_factor

MA77_factor performs the numerical factorization. On the call to MA77_factor, the user must specify whether or not the matrix is positive definite. If pos_def is set to .true., no pivoting is performed and, if a non-positive pivot is encountered, the computation will terminate with the error flag set to -11.

The factorization uses the assembly tree and the ordering of the children that was set up by MA77_analyse. Starting at a root node (one that has no parent), a subroutine that recursively factorizes the children of the root and its descendants is called. In the positive definite case, all the data structures are set up before the factorization begins and, at each stage of the factorization, the partial factorization of the frontal matrix is performed by HSL_MA54. In the indefinite case, the partial factorizations are performed by HSL_MA64. The relative pivot tolerance u is initially set to control%u. If a pivot candidate does not satisfy the threshold pivot criteria, the action taken depends on the control parameters control%umin and control%static. If static pivoting is not requested (control%static=0.0) and control%umin=control%u, the pivot is delayed (this is the default case). If control%umin<control%u and the relative pivot tolerance for the pivot candidate is $v \geq$control%umin, the candidate is accepted and u is set to v. The new value is used in subsequent calls to HSL_MA64. If $v <$control%umin, the pivot is delayed unless static pivoting is being used. In this case, if the candidate has absolute value at least control%static, it is selected and info%num_nothresh is incremented by one; otherwise, the pivot is given the value that has the same sign but absolute value control%static and info%num_perturbed is incremented by one. Further details are provided in the documentation for HSL_MA64. Note that if a small relative pivot tolerance is used and/or static pivoting is used, the factorization is likely to be inaccurate and an iterative procedure (such as iterative refinement) may be needed once the factorization is complete to try and restore accuracy. Our experience is that the accuracy can be very sensitive to the choice of control%static; in our tests in double precision, a value of $10^{-6}\|\mathbf{A}\|$ was an appropriate choice.

The real data for delayed pivots is held in the superfile identified by filename(4). Delayed pivots mean that the arrays set up at the start of the factorization, including the array that holds the frontal matrix, may not be large enough and may have to be reallocated to allow the computation to continue. The original size of these arrays and the amount by which they are increased when reallocated is controlled by control%multiplier.

If the user passes right-hand vectors to MA77_factor_solve, the forward substitutions are performed as the factor entries are generated, by calling MA54_solve and MA64_solve for the positive definite and indefinite cases, respectively. Once the factorization is complete, the back substitutions are performed by calling MA77_solve with job = 3 (positive definite case) or job = 2 followed by job = 3 (indefinite case).

MA77_solve
Having checked the user's data, MA77_solve performs a number of steps: forward substitution followed by a diagonal solve (indefinite case only), followed by back substitution (unless only one of these is requested). Each step starts at a root node and, for each of its children, calls a subroutine recursively. This in turn calls MA54_solve or, in the indefinite case, MA64_solve. The matrix factor must be accessed once for the forward substitution and once for the back substitution. This is independent of the number of right-hand sides so that solving for several right-hand sides at once is significantly faster than repeatedly solving for a single right-hand side.

References:

[1] J.K Reid and J.A. Scott. (2009). An out-of-core Cholesky solver. *ACM Transactions on Mathematical Software*, 36, Article 9.

[2] D.A. Ruiz. (2001). A scaling algorithm to equilibrate both row and column norms in matrices. RAL Technical Report. RAL-TR-2001-034.

[3] J.A. Scott. (2008). Scaling and pivoting in an out-of-core sparse direct solver. RAL Technical Report. RAL-TR-2008-016.

[4] J.K Reid and J.A. Scott. (2008). An efficient out-of-core sparse symmetric indefinite direct solver. Technical Report TR-RAL-2008-024.

5 EXAMPLE OF USE

We give an example of the code required to solve a set of equations using HSL_MA77 when input is by rows (example 5.1) and when input is by elements (example 5.2).

5.1 Row entry

Suppose we wish to factorize the matrix

$$A = \begin{pmatrix} 2. & 3. & & & \\ 3. & 1. & 4. & & 6. \\ & 4. & 1. & 5. & \\ & & 5. & 3. & \\ & 6. & & & 1. \end{pmatrix}$$

and then solve for the right-hand side

$$B = \begin{pmatrix} 5. \\ 14. \\ 10. \\ 8. \\ 7. \end{pmatrix}$$

and compute the residuals. The following code may be used. Note that, in this example, it would be more efficient to pass the right-hand side to `MA77_factor`; here our aim is to illustrate calling `MA77_solve` after `MA77_factor`.

```
program example1
! Simple code to illustrate row entry to hsl_ma77
      use hsl_ma77_double
      implicit none

! Derived types
      type (ma77_keep)    :: keep
      type (ma77_control) :: control
      type (ma77_info)    :: info

! Parameters
      integer, parameter :: wp = kind(0.0d0)
      integer, parameter :: mvar = 10 ! largest number of entries in a row

      integer, dimension (:),    allocatable :: order
      real(wp), dimension (:,:), allocatable :: x,resid

      integer  :: index(mvar)
      real(wp) :: values(mvar)
      character(len=20) :: filename(4)
      integer :: lx,lresid,n,name,nrhs,nvar
      logical :: pos_def

! Read in the order n of the matrix
      read (*,*) n
! Choose file indentifiers (hold direct access files in current directory)
      filename(1) = 'factor_integer'
      filename(2) = 'factor_real'
      filename(3) = 'work_real'
      filename(4) = 'temp1'
```

```
! Allocate arrays of appropriate size
      allocate (order(n),x(1:n,1:1),resid(1:n,1:1))

! Initialisation
      call ma77_open(n,filename,keep,control,info)
      if (info%flag < 0) go to 100

! For each row of the matrix, read in the number of entries, the
! column indices and numerical values. Then call ma77_input_vars and
! ma77_input_reals to enter the integer and real data for the row.
      do name = 1,n
        read (*,*) nvar
        read (*,*) index(1:nvar)
        read (*,*) values(1:nvar)
        call ma77_input_vars(name,nvar,index,keep,control,info)
        if (info%flag < 0) go to 100
        call ma77_input_reals(name,nvar,values,keep,control,info)
        if (info%flag < 0) go to 100
      end do

! Use the natural pivot order 1,2,...,n
      do name = 1,n
        order(name) = name
      end do

! Perform analyse and factorise
      call ma77_analyse(order,keep,control,info)
      if (info%flag < 0) go to 100
      pos_def = .false.
      call ma77_factor(pos_def,keep,control,info)
      if (info%flag < 0) go to 100

! Read in the right-hand side and copy into resid.
      read (*,*) x(1:n,1:1)
      resid(1:n,1:1) = x(1:n,1:1)
! Solve and then compute the residuals
      nrhs = 1
      lx = size(x,1)
      call ma77_solve(nrhs,lx,x,keep,control,info)
      if (info%flag < 0) go to 100
      lresid = size(resid,1)
      call ma77_resid(nrhs,lx,x,lresid,resid,keep,control,info)
      if (info%flag < 0) go to 100
      write (*,'(/a,/,(6f10.3))') ' The computed solution is:',x(1:n,1:1)
      write (*,'(/a,/,(6f10.3))') ' The residuals are:',resid(1:n,1:1)

 100  call ma77_finalise(keep,control,info)

! Deallocate all arrays
```

```
      deallocate (order,x,resid)
end program example1
```

with the following data:

```
   5
   2
   1  2
   2. 3.
   4
   1  2  3  5
   3. 1. 4. 6.
   3
   2  3  4
   4. 1. 5.
   2
   3  4
   5. 3.
   2
   2  5
   6. 1.
   5. 14. 10. 8. 7.
```

This produces the following output:

```
 The computed solution is:
     1.000     1.000     1.000     1.000     1.000

 The residuals are:
     0.000     0.000     0.000     0.000     0.000
```

5.2 Element entry

To illustrate the element entry, assume we wish to solve a problem comprising the following four elemental matrices $\mathbf{A}^{(\mathbf{k})}$, $1 \le k \le 4$:

$$\begin{array}{c} 2 \\ 5 \end{array} \begin{pmatrix} 2 & 1 \\ 1 & 7 \end{pmatrix} \quad \begin{array}{c} 5 \\ 8 \end{array} \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} \quad \begin{array}{c} 2 \\ 5 \\ 3 \\ 6 \end{array} \begin{pmatrix} 4 & 3 & 2 & 3 \\ 3 & 10 & 3 & 2 \\ 2 & 3 & 6 & 1 \\ 3 & 2 & 1 & 6 \end{pmatrix} \quad \begin{array}{c} 5 \\ 8 \\ 6 \\ 9 \end{array} \begin{pmatrix} 4 & 1 & 1 & 3 \\ 1 & 3 & 2 & 2 \\ 1 & 2 & 2 & 1 \\ 3 & 2 & 1 & 4 \end{pmatrix}$$

where the variable indices are indicated by the integers before each matrix. We note that, in this example, not all the integers in the range $1 \le i \le n$ ($n = 9$) are used to index a variable. The following program may be used to solve this problem. For each element we read the integer and real data into arrays `eltvar` and `values`. More than one call to `MA77_input_reals` is used to enter the real data. The solution phase is performed at the same time as the factorization by calling `MA77_factor_solve`.

```
program example2
! Simple code to illustrate element entry to hsl_ma77
      use hsl_ma77_double
      implicit none

! Derived types
      type (ma77_keep)    :: keep
      type (ma77_control) :: control
      type (ma77_info)    :: info

! Parameters
      integer, parameter :: wp = kind(0.0d0)
      integer, parameter :: mvar = 4 ! largest number of variables in an element

      integer, dimension (:),    allocatable :: order
      real(wp), dimension (:,:), allocatable :: x,resid

      integer  :: eltvar(mvar)
      real(wp) :: values((mvar*mvar+mvar)/2)
      character(len=20) :: filename(4)
      integer :: i,ielt,irhs,k1,k2,lx,lresid,n,nelt,nrhs,nvar
      logical :: pos_def

! Read in the order n, the number of elements and number of right-hand sides
      read (*,*) n,nelt,nrhs
! Choose file identifiers (hold files in current directory)
      filename(1) = 'factor_integer'
      filename(2) = 'factor_real'
      filename(3) = 'work_real'
      filename(4) = 'temp1'

! Allocate arrays of appropriate size
      allocate (order(n),x(1:n,1:nrhs),resid(1:n,1:nrhs))

! Initialisation
      call ma77_open(n,filename,keep,control,info,nelt=nelt)
      if (info%flag < 0) go to 100

! For each element, read in the number of variables, the
! variable indices and numerical values (lower triangular part of element).
      do ielt = 1,nelt
        read (*,*) nvar
        read (*,*) eltvar(1:nvar)
        read (*,*) values(1:(nvar*nvar+nvar)/2)
        call ma77_input_vars(ielt,nvar,eltvar,keep,control,info)
        if (info%flag < 0) go to 100
! To illustrate entering reals using more than one call to ma77_input_reals,
! we enter the reals of the element matrix one column at a time.
        k1 = 1
        do i = 1,nvar
          k2 = k1 + nvar - i
          call ma77_input_reals(ielt,nvar-i+1,values(k1:k2),keep,control,info)
          if (info%flag < 0) go to 100
          k1 = k2 + 1
        end do
      end do

! Use the natural pivot order 1,2,...,n
      do i = 1,n
        order(i) = i
      end do

! Perform analyse
      call ma77_analyse(order,keep,control,info)
      if (info%flag < 0) go to 100

! Read in the right-hand sides and copy into resid.
      do irhs = 1,nrhs
        read (*,*) x(1:n,irhs)
      end do
      resid(1:n,1:nrhs) = x(1:n,1:nrhs)

! Perform factorisation and solve together
      pos_def = .true.
      lx = size(x,1)
      call ma77_factor_solve(pos_def,keep,control,info,nrhs,lx,x)
      if (info%flag < 0) go to 100

! Compute the residuals
      lresid = size(resid,1)
      call ma77_resid(nrhs,lx,x,lresid,resid(1:n,1:nrhs),keep,control,info)
      if (info%flag < 0) go to 100
      do irhs = 1,nrhs
        write (*,'(/a,i2)') ' For right-hand side ',irhs
        write (*,'(/a,/,(6f10.3))') ' The computed solution is:',x(1:n,irhs)
        write (*,'(/a,/,(6f10.3))') ' The residuals are:',resid(1:n,irhs)
      end do

 100  call ma77_finalise(keep,control,info)

! Deallocate all arrays
      deallocate (order,x,resid)
end program example2
```

To solve for the right-hand sides

$$\mathbf{B} = \begin{pmatrix} 0 & 0 \\ 0 & 15 \\ -7 & 12 \\ 0 & 0 \\ -20 & 40 \\ 0 & 18 \\ 0 & 0 \\ -1 & 14 \\ 1 & 10 \end{pmatrix}$$

the required input data is:

```
  9  4  2
  2
  2  5
  2. 1. 7.
  2
  5  8
  3. 2. 4.
  4
  2  5  3  6
  4. 3. 2. 3. 10. 3. 2. 6. 1. 6.
  4
  5  8  6  9
  4. 1. 1. 3. 3. 2. 2. 2. 1. 4.
  0.  0.  -7.  0. -20.  0. 0. -1.  1.
  0. 15.  12.  0.  40. 18. 0. 14. 10.
```

This produces the following output:

```
 For right-hand side  1

 The computed solution is:
     0.000     1.000    -1.000     0.000    -1.000     0.000
     0.000     0.000     1.000

 The residuals are:
     0.000     0.000     0.000     0.000     0.000     0.000
     0.000     0.000     0.000

 For right-hand side  2

 The computed solution is:
     0.000     1.000     1.000     0.000     1.000     1.000
     0.000     1.000     1.000

 The residuals are:
     0.000     0.000     0.000     0.000     0.000     0.000
     0.000     0.000     0.000
```