



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 k th 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 2.5.5 and 2.5.7 (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 2.5.19). 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 $\bar{A} = SAS$ is computed, where S is a diagonal scaling matrix.

ATTRIBUTES — **Version:** 6.5.0 (04 July 2025). **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 + C interoperability). **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 C interface to Fortran code

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

The wrapper will automatically convert between 0-based (C) and 1-based (Fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing. The conversion may be disabled by setting the control parameter `control.f_arrays=1` and supplying all data using 1-based indexing. With 0-based indexing, the matrix is treated as have rows and column $0, 1, \dots, n-1$. In this document, we assume 0-based indexing.

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

2.2 OpenMP

OpenMP is used by HSL_MA77 to provide parallelism for shared memory environments. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of `-openmp`). The number of threads may be controlled at runtime by setting the environment variable `OMP_NUM_THREADS`.

2.3 Calling sequences

Access to the package requires inclusion of the header file

Single precision version

```
#include "hsl_ma77s.h"
```

Double precision version

```
#include "hsl_ma77d.h"
```

It is not possible to use more than one module at the same time.

The following procedures are available to the user:

- `ma77_default_control` may be called to give members of `struct ma77_control` their default values.
- `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 \mathbf{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 \mathbf{A} have been input and after the call to `ma77_analyse`. The matrix \mathbf{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 \mathbf{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.

2.4 The derived data types

For each problem, the user must employ the derived types defined in the header file to declare scalars of the types `struct ma77_control`, `struct ma77_info`, and a `void *` pointer `keep`. The following pseudocode illustrates this.

```
#include "hsl_ma77d.h"
...
struct ma77_control control;
struct ma77_info info;
void *keep;
...
```

The components of `ma77_control` and `ma77_info` are explained in Sections 2.5.19 and 2.5.20. The `void *` pointer is used to pass data between the subroutines of the package.

2.5 Argument lists and calling sequences

2.5.1 Real kinds

The descriptions we give are for the double precision version of the package. In the single precision version `double` is replaced by `float` throughout.

2.5.2 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 floating point array that can be allocated is taken to be the Fortran value `INT_MAX/4` in the single precision version and `INT_MAX/8` in the double precision version, where `INT_MAX` is the largest integer representable in type `int`. On a 64-bit architecture, it is taken to be `LONG_MAX/4` and `LONG_MAX/8`, respectively, where `LONG_MAX` is the largest integer representable in type `long`.

2.5.3 The default setting subroutine

Default values for components of the `ma77_control` structure may be set a call to `ma77_default_control`.

```
void ma77_default_control(struct ma77_control *control)
```

`control` has its components set to their default values, as described in Section 2.5.19.

2.5.4 The initialization subroutine

Data structures are set up and superfiles are opened by a call to `ma77_open` (for assembled problems) or `ma77_open_nelt` (for element problems).

```
void ma77_open(const int n, const char* fname1, const char* fname2,
              const char *fname3, const char *fname4, void **keep,
              const struct ma77_control *control, struct ma77_info *info)
```

```
void ma77_open_nelt(const int n, const char* fname1, const char* fname2,
                   const char *fname3, const char *fname4, void **keep,
                   const struct ma77_control *control, struct ma77_info *info,
                   const int nelt)
```

`n` must be set to the matrix order. **Restriction:** $n \geq 0$.

`fname1` is a string with at most 400 characters. It specifies the filename used to hold the integer matrix and factor data. It must be distinct from `fname2`, `fname3` and `fname4`. This must be provided even if the user wishes to run in-core. **Restriction:** `strlen(fname1) ≤ 400`

`fname2` is a string with at most 400 characters. It specifies the filename used to hold the floating-point matrix and factor data. It must be distinct from `fname1`, `fname3` and `fname4`. This must be provided even if the user wishes to run in-core. **Restriction:** `strlen(fname2) ≤ 400`

`fname3` is a string with at most 400 characters. It specifies the filename used to hold the floating-point work space. It must be distinct from `fname1`, `fname2` and `fname4`. This must be provided even if the user wishes to run in-core. **Restriction:** `strlen(fname3) ≤ 400`

`fname4` is a string with at most 400 characters. It specifies the filename used to hold additional floating-point work space in the indefinite case. It must be distinct from `fname1`, `fname2` and `fname3`. This must be provided even if the user wishes to run in-core, or solve only positive definite problems. **Restriction:** `strlen(fname4) ≤ 400`

`keep` will be set to point at an area of memory used to hold data about the problem being solved and must be passed unchanged to the other subroutines. To avoid a memory leak the subroutine `ma77_finalise` must be called to clean up and deallocate this memory once the data is no longer required.

`control` is used to control the actions of the package, see Section 2.5.19.

`info` is used to return information about the execution of the package, as explained in Section 2.5.20.

`nelt` is used when input is by elements. If input is by elements, `nelt` must be set to be at least the largest integer used to index an element. **Restriction:** `nelt ≥ 0`.

2.5.5 The input of integer data

A call to the following routine must be made for each element or row:

```
void ma77_input_vars(const int index, const int nvar, const int list[],
    void **keep, const struct ma77_control *control, struct ma77_info *info)
```

`index` must hold the index of the incoming element or row (numbering starts at one). 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` must hold the number of variables in the incoming element or row. **Restriction:** `nvar ≥ 0`.

`list` is an array of size at least `nvar`. It must hold the indices of the variables in the incoming element or row. Duplicates are allowed. Out-of-range indices are ignored.

`keep` must be passed unchanged by the user.

`control` is used to control the actions of the package, see Section 2.5.19.

`info` provides information about the execution of the subroutine, as explained in Section 2.5.20. It must be passed unchanged by the user.

2.5.6 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 routine must be made:

```
void ma77_analyse(const int order[], void **keep,
    const struct ma77_control *control, struct ma77_info *info);
```

`order` is an array of size at least n . It must specify the elimination order. If i is used to index a variable, $\text{abs}(\text{order}[i])$ must hold its position in the pivot sequence. If a 1×1 pivot i is required, the user must set $\text{order}[i] > 0$. If a 2×2 pivot involving variables i and j is required, the user must set $\text{order}[i] < 0$, $\text{order}[j] < 0$ and $|\text{order}[j]| = |\text{order}[i]| + 1$. If i , $0 \leq i \leq n-1$, is not used to index a variable, $\text{order}[i]$ may have any value and this is replaced by zero. Note that unless Fortran array numbering is used this means that the first pivot cannot be marked as part of a 2×2 pivot. 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 \mathbf{A} is **not** positive definite.

`keep`, `control`, `info`: see Section 2.5.5.

2.5.7 The input of floating point 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.

```
void ma77_input_reals(const int index, const int length,
    const double reals[], void **keep, const struct ma77_control *control,
    struct ma77_info *info);
```

`index` must hold the index of the incoming element or row. If `index` is out-of-range, the element or row is ignored.

`length` must hold the number of reals being input on this call. **Restriction:** $\text{length} \geq 0$.

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

`keep`, `control`, `info`: see Section 2.5.5.

2.5.8 To compute scaling factors

To compute a scaling of the matrix \mathbf{A} , a call to the following routine may be made after the calls to `ma77_input_reals` are complete and after the call to `ma77_analyse`:

```
void ma77_scale(double scale[], void **keep,
    const struct ma77_control *control, struct ma77_info *info,
    double *anorm);
```

`scale` is an array of size at least n . On exit, the first n entries contains the diagonal entries of the scaling matrix \mathbf{S} .

`keep`, `control`, `info`: see Section 2.5.5.

`anorm` may be null. If it is non-null, then on exit, it holds $\|\mathbf{A}\|_\infty$ (regardless of the value of `control.infnorm`).

2.5.9 To factorize the matrix and optionally solve $AX = B$

To factorize the matrix, a call to the following routine must be made after the calls to `ma77_input_reals` are complete:

```
void ma77_factor(const int posdef, void **keep,
                const struct ma77_control *control, struct ma77_info *info,
                const double *scale)
```

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

```
void ma77_factor_solve(const int posdef, void **keep,
                      const struct ma77_control *control, struct ma77_info *info, const double *scale,
                      const int nrhs, const int lx, double rhs[])
```

`pos_def` should be set to a non-zero (true) value if **A** is known to be positive definite (ignoring any unused rows and columns) and to 0 (false) otherwise. If `pos_def != 0`, no numerical pivoting for stability is performed during the numerical factorization.

`keep`, `control`, `info`: see Section 2.5.5.

`nrhs` holds the number of right-hand sides. **Restriction:** `nrhs` ≥ 1 .

`lx` must be set to the first extent of the array `x`. **Restriction:** `lx` $\geq n$.

`x` is a column major rank-2 array with extents `lx` and `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[(j-1)*lx+i-1]` holds the component of the right-hand side for variable `i` to the `j`th system. On exit, `x[(j-1)*lx+i-1]` holds the solution for variable `i` to the `j`th system.

`scale` may be NULL. If it is not NULL it must be a rank-1 array of size at least `n`, and contain the diagonal entries of the scaling matrix **S**.

2.5.10 To solve linear systems using the computed factors

After the call to `ma77_factor`, one or more calls to the following routine may be made to solve $AX = B$. Partial solutions may be performed by appropriately setting the optional parameter `job`.

```
void ma77_solve(const int job, const int nrhs, const int lx, double x[],
                void **keep, const struct ma77_control *control, struct ma77_info *info,
                const double *scale)
```

`job` If `job=0`, $AX = B$ is solved. In the positive-definite case, the Cholesky factorization that has been computed may be expressed in the form

$$SAS = (PL)(PL)^T$$

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

$$SAS = (PL)D(PL)^T$$

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 not NULL 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:

- 0 for solving $\mathbf{AX} = \mathbf{B}$
- 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: $\text{job} = 0, 1, 2, 3, 4$.

`nrhs` holds the number of right-hand sides. **Restriction:** $\text{nrhs} \geq 1$.

`lx` holds the first extent of `x`. **Restriction:** $\text{lx} \geq n$.

`x` is a column major rank-2 array with extents `lx` and `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[(j-1)*lx+i-1]` holds the component of the right-hand side for variable `i` to the `j`th system. On exit, `x[(j-1)*lx+i-1]` holds the solution for variable `i` to the `j`th system.

`keep`, `control`, `info`: see Section 2.5.5.

`scale` may be NULL. Otherwise it must be a rank-1 array of size at least `n`. If `scale` was non-NULL on the last call to `ma77_factor` (or `ma77_factor_solve`), it must also be non-NULL on each call to `ma77_solve` and `scale(1:n)` must be unchanged since the call to `ma77_factor` (or `ma77_factor_solve`).

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

Following a call to `ma77_factor_solve` or to `ma77_solve` (with `job=0`), the residual matrix $\mathbf{B} - \mathbf{AX}$ may be computed by making a call to:

```
void ma77_resid(const int nrhs, const int lx, const double x[],
               const int lresid, double resid[], void **keep,
               const struct ma77_control *control, struct ma77_info *info,
               double *anorm_bnd);
```

`nrhs` holds the number of right-hand sides for which the residuals are required. **Restriction:** $\text{nrhs} \geq 1$.

`lx` holds the first extent of `x`. **Restriction:** $\text{lx} \geq n$.

`x` is a column-major rank-2 array with extents `lxb` and `nrhs`. It must be set to hold the matrix \mathbf{X} .

`lresid` holds the first extent of `resid`. **Restriction:** $\text{lresid} \geq n$.

`resid` is a column-major rank-2 array with extents `lresid` and `nrhs`. 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 2.5.5.

`anorm_bnd` may be null. If it is non-null, then on exit, it holds $\|\mathbf{A}\|_\infty$ in the row-entry case and $\sum_{k=1}^m \|\mathbf{A}^{(k)}\|_\infty$ in the element case, which is an upper bound for $\|\mathbf{A}\|_\infty$.

2.5.12 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 (`ma77_finalise_restart` only), a call of the following form should be made to free memory pointed to by `keep` and to close the files opened by the package for the problem:

```
void ma77_finalise(void **keep, const struct ma77_control *control,
                  struct ma77_info *info)

void ma77_finalise_restart(void **keep, const struct ma77_control *control,
                           struct ma77_info *info, char *restart_file)
```

`keep` must be passed unchanged. On exit, all memory associated with `keep` will have been freed.

`control` will be used to control printing. Only the components that control printing are accessed (see Section 2.5.19).

`info` provides information about the execution of the subroutine, as explained in Section 2.5.20.

`restart_file` must have length at most 500 characters. 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 `fname1` and `fname2` are saved and those identified by `fname3` and `fname4` (see Section 2.5.4) are deleted. **Restriction:** `strlen(restart_file) ≤ 500`.

2.5.13 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 to the following routine should be made:

```
void ma77_restart(const char *restart_file, const char *fname1,
                  const char *fname2, const char *fname3, const char *fname4, void **keep,
                  const struct ma77_control *control, struct ma77_info *info)
```

`restart_file` has length at most 500 characters. It must hold the name of the sequential access file that contains the data that was written by a call to `ma77_finalise_restart`. The file must not have been changed since then. On successful exit, this file will be closed and deleted. **Restriction:** `strlen(restart_file) ≤ 500`.

`fname1`, `fname2`, `fname3` and `fname4` must length at most 400 characters. They hold identifiers for the superfiles. The data in the primary files that are identified by `fname1` and `fname2` and their secondaries, if any, must be unchanged since the call to `ma77_finalise`. There must be no files with names `fname3` or `fname4`. **Restriction:** `strlen(fname1) ≤ 400`, `strlen(fname2) ≤ 400`, `strlen(fname3) ≤ 400`, `strlen(fname4) ≤ 400`.

`keep` will have been initialised and information required by `ma77_solve` or `ma77_factor` or `ma77_factor_solve` will have been allocated and the contents restored.

`control` used to control printing. Only the components that control printing are accessed (see Section 2.5.19).

`info` provides information about the execution of the subroutine, as explained in Section 2.5.20.

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

After a successful call to `ma77_factor` or to `ma77_factor_solve` with `posdef != 0` (true), information on the pivots may be obtained using a call to

```
void ma77_enquire_posdef(double d[], void **keep,
    const struct ma77_control *control, struct ma77_info *info)
```

`d` is an array of size n . The i -th pivot will be placed in `d[i]`, $i = 0, 2, \dots, n-1$.

`keep`, `control`, `info`: see Section 2.5.5.

2.5.15 To obtain information on the factorization (indefinite case)

After a successful call to `ma77_factor` or to `ma77_factor_solve` with `posdef == 0` (false), information on the pivot sequence and the matrix \mathbf{D}^{-1} may be obtained using a call to

```
void ma77_enquire_indef(int *piv_order, double *d, void **keep,
    const struct ma77_control *control, struct ma77_info *info)
```

`piv_order` is an array 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 , $1 \leq i \leq n$, is not used to index a variable, `piv_order[i]` will be set to zero.

`d` is a column-major rank-2 array with extents 2 and n . The diagonal entries of \mathbf{D}^{-1} will be placed in `d[(i-1)*2]`, $i = 1, 2, \dots, n$, the off-diagonal entries of \mathbf{D}^{-1} will be placed in `d[(i-1)*2+1]`, $i = 1, 2, \dots, n-1$, and `d[(n-1)*2+1]` will be set to zero.

`keep`, `control`, `info`: see Section 2.5.5.

2.5.16 To alter \mathbf{D}^{-1}

After a successful call to `ma77_factor` or to `ma77_factor_solve` with `posdef == 0` (false), the matrix \mathbf{D}^{-1} may be altered using to

```
void ma77_alter(const double d[], void **keep,
    const struct ma77_control *control, struct ma77_info *info)
```

`d` is a column-major rank-2 array with extents 2 and n . The diagonal entries of \mathbf{D}^{-1} will be altered to `d[(i-1)*2]`, $i = 1, 2, \dots, n$, and the off-diagonal entries will be altered to `d[(i-1)*2+1]`, $i = 1, 2, \dots, n-1$ (and the factorization of \mathbf{A} will no longer be available).

`keep`, `control`, `info`: see Section 2.5.5.

2.5.17 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 $\mathbf{A}\mathbf{Y}_j = 0$ and $\mathbf{Y}_j^T \mathbf{B}_j \neq 0$.

```
void ma77_solve_fredholm(int nrhs, int flag_out[], int lx, double x[],
    void **keep, const struct ma77_control *control,
    struct ma77_info *info, const double *scale);
```

`nrhs`, `lx`, `keep`, `control`, `info`, `scale`: see Section 2.5.10.

`flag_out` is a rank-1 array of size `nrhs`. On exit, `flag_out[j]` is set to 0 (i.e. evaluates to true) if the j -th system is consistent and to 1 (i.e. false) otherwise.

`x` is a rank-2 array 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[j*lx+i]` holds the component of the right-hand side for variable i to the j -th system ($j=0, 1, \dots, nrhs-1$). On exit, the first $nrhs*lx$ elements hold the same solution as is returned by `ma77_solve` and, if `flag_out(j)=0`, the n elements starting at `x[(nrhs+j)*lx]` hold the Fredholm alternative solution for the j -th system.

2.5.18 To form a matrix product with PL or $(PL)^T$

After a call to `ma77_factor` or `ma77_factor_solve`, the product $Y = PLX$ or $Y = (PL)^T X$ (or $Y = S^{-1}PLX$ or $Y = (S^{-1}PL)^T X$ if `scale` was present on the call to `ma77_factor` or `ma77_factor_solve`) may be computed using a call of the following form

```
void ma77_multiply(int trans, int k, int lx, double x[], int ly,
                 double y[], void **keep, const struct ma77_control *control,
                 struct ma77_info *info, const double *scale);
```

`trans` determines the system to solve. If `trans` $\neq 0$ (i.e. it evaluates to true), $Y = (PL)^T X$ (or $Y = (S^{-1}PL)^T X$) is computed; otherwise, $Y = PLX$ (or $Y = S^{-1}PLX$) is computed.

`k` holds the number of columns of X and Y . **Restriction:** $k \geq 1$.

`x` is a rank-2 array with extents `lx` and `k`. It must be set to hold the matrix X . It is altered only if `trans` $\neq 0$ (i.e. is true) and `scale` is not NULL (in this case, `x` holds $S^{-1}X$ on exit).

`y` is a rank-2 array with extents `ly` and `k`. On exit, it holds the requested matrix product. Note that if variable i is not used to index a variable, `y[j*ly+i]` is set to zero ($j=0, 1, \dots, nrhs-1$).

`keep`, `control`, `info`, `scale`: see Section 2.5.10.

2.5.19 The derived data type for holding control parameters

The derived data type `ma77_control` is used to hold controlling data. The components, and default values that are set by a call to `ma77_default_control`, are:

C only controls

`int f_arrays` indicates whether to use C or Fortran array indexing. If `f_arrays` $\neq 0$ (i.e. evaluates to true) then 1-based indexing of the arrays `list`, `piv_order` and the scalar `idx` is assumed. Otherwise, if `f_arrays` $= 0$ (i.e. evaluates to false), then these arrays are copied and converted to 1-based indexing in the wrapper function. All descriptions in this documentation assume `f_arrays` $= 0$. The default is `f_arrays` $= 0$ (false).

Printing controls

`int print_level` 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`.

`int unit_diagnostics` holds the Fortran unit number for diagnostic printing. Printing is suppressed if `unit_diagnostics < 0`. The default is `unit_diagnostics=6` (stdout).

`int unit_error` holds the unit number for error messages. Printing of error messages is suppressed if `unit_error < 0`. The default is `unit_error=6` (stdout).

`int unit_warning` holds the unit number for warning messages. Printing of warning messages is suppressed if `unit_warning < 0`. The default is `unit_warning=6` (stdout).

Controls used by `ma77_open`

`int bits` 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.

`int buffer_lpage[2]` 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[0] = buffer_lpage[1] = 212`.
Restriction: $1 \leq \text{buffer_lpage}[i] \leq \text{file_size}$, $i=0$ or 1 .

`int buffer_npage[2]` holds the number of pages in the integer and reals in-core buffers that are used by HSL_OF01. The default is `buffer_npage[0] = buffer_npage[1] = 1600`. **Restriction:** $\text{buffer_npage}[i] \geq 1$, $i=0$ or 1

`long int file_size` 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.

`long int maxstore` 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 2.7. The default is `maxstore=0`. **Restriction:** $\text{maxstore} \geq 0$. When running on 32-bit architecture (`control.bits=32`), the value of `maxstore` must not exceed `INT.MAX/4` in the single precision version and `INT.MAX/8` in the double precision version; on a 64-bit architecture, `maxstore` must not exceed `LONG.MAX/4` in the single precision version and `LONG.MAX/8` in the double precision version.

`long int storage[3]` controls in-core working. If the user wants to use arrays in place of the superfiles `fname1`, `fname2` or `fname3`, then `storage[0]`, `storage[1]` or `storage[2]` should be set respectively 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 `fname1`, `fname2`, `fname3` is used ($i = 0, 1, 2$). Further details are given in Section 2.7. The default is `storage[i]=0`, $i=0, 1, 2$

Controls used by `ma77_analyse`

`int nemin` 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`

`int maxit` specifies the maximum number of iterations performed by the scaling algorithm. The default is `maxit=1`. The default is used if `maxit<1`.

`int infnorm` 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`.

`double thresh` has 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 $1 \pm \text{thresh}$.

Controls used by `ma77_factor` with `pos_def!=0` (true)

`int nb54` 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==0` (false)

`int action` controls behaviours in case of singularity. 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 a non-zero value (true) or terminates (see error -11) if it is 0 (false). The default is `action=1`.

`double multiplier` controls memory management behaviour. To allow for delayed pivots, the arrays that hold the frontal matrix and its index list are allocated to accommodate a matrix of order $s \times \max(1, \text{multiplier})$ 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 $s_c \times \max(1, \text{multiplier})$, where s_c is the current front size. The default value is 1.1.

`int nb64` 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 $\text{mod}(\text{nb64}, \text{nbi})=0$. If this restriction is not satisfied, the block size is taken to be $\max(\text{nbi}, \text{nb64}/\text{nbi} * \text{nbi})$.

`int nbi` 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`.

`double small` controls pivoting. Any pivot whose modulus is less than `small` is treated as zero. The default is `small = 1 \times 10^{-20}`.

`double static` 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 \geq small`.

`int storage_indef` controls use of in-core memory. If the user wants to use arrays in place of the superfile `fname4`, `storage_indef` should be set to the initial size for this arrays; if the size is zero, guesses based on the value of the component `maxstore` will be made. If the array is found to be not large enough, the superfile may be used instead. `storage_indef` is not accessed if `maxstore=0`. If `storage_indef<0`, a superfile identified by `fname4` is used. Further details are given in Section 2.7. The default is `storage_indef=0`.

double `u` 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.

double `umin` 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 $v \geq u_{\min}$, 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 $u_{\min}=1.0$. 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`

double `consist_tol` has default value of machine epsilon (as returned by the Fortran intrinsic `epsilon()`). During the solve, a tolerance equal to `consist_tol * n * ||Bi||` is used to determine whether the system of equations with the right-hand side \mathbf{B}_i is consistent. It is only accessed if \mathbf{A} is singular.

2.5.20 The derived data type for holding information

The derived data type `struct 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:

double `detlog` holds, on exit from `ma77_factor` or `ma77_factor_solve`, the logarithm of the absolute value of the determinant of \mathbf{A} or zero if the determinant is zero.

int `detsign` holds, on exit from `ma77_factor` or `ma77_factor_solve`, the sign of the determinant of \mathbf{A} or zero if the determinant is zero.

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

int `index[4]` holds, on exit from `ma77_open` and `ma77_restart`, `index[i]` the HSL_OF01 index of the superfile identified by `fname1`, `fname2`, `fname3`, `fname4` in `index[i]` ($i=0, 1, 2, 3$). 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.

int `iostat` holds the Fortran `iostat` parameter.

int `matrix_dup` is set, on each exit from `ma77_input_vars` to the total number of duplicate entries that have been found.

int `matrix_outrange` is set, on each exit from `ma77_input_vars` to the total number of out-of-range entries that have been found.

int `matrix_rank` is set, on exit from `ma77_factor` or `ma77_factor_solve`, to hold the computed rank of the factorized matrix.

int `maxdepth` holds, on exit from `ma77_analyse`, the maximum depth of the assembly tree.

int `maxfront` holds, on exit from `ma77_analyse`, 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.

long int `minstore` holds, on exit from `ma77_factor` or `ma77_factor_solve`, 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.

`int ndelay` holds, on exit from `ma77_factor` or `ma77_factor_solve`, 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.

`long int nfactor` holds, on exit from `ma77_analyse`, 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.

`long int nflops` holds, on exit from `ma77_analyse`, 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.

`long int nio_read[2]` holds, on exit from a call to `ma77_analyse`, `ma77_factor`, `ma77_factor_solve`, and `ma77_solve`, the number of integer (`nio_read[0]`) and real (`nio_read[1]`) records actually read from disk by HSL_OF01 during the subroutine call.

`long int nio_write[2]` holds, on exit from a call to `ma77_analyse`, `ma77_factor`, `ma77_factor_solve`, and `ma77_solve`, the number of integer (`nio_write[0]`) and real (`nio_write[1]`) records actually written to disk by HSL_OF01 during the subroutine call.

`int niter` holds, on exit for `ma77_scale`, the number of iterations of the scaling algorithm that were performed.

`int nsup` holds, on exit from the final call to `ma77_input_vars`, the number of supervariables in the problem (see Section 4).

`int ntwo` holds, on exit from `ma77_analyse`, 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**.

`int num_file[4]` holds, on exit from a call to `ma77_finalise`, the number of secondary files used by each of the superfiles.

`int num_neg` holds, on exit from `ma77_factor` or `ma77_factor_solve`, the number of negative eigenvalues of the matrix **D**.

`int num_nothresh` holds, on successful exit from `ma77_factor` or `ma77_factor_solve`, 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`.

`int num_perturbed` holds, on successful exit from `ma77_factor` or `ma77_factor_solve`, the number of pivots that were perturbed to `control.static` or `-control.static`. It will have the value zero if `control.static=0.0`.

`long int nwd_read[2]` holds, on exit from a call to `ma77_analyse`, `ma77_factor`, `ma77_factor_solve`, and `ma77_solve`, the number of integer and real scalars read by HSL_OF01 during the subroutine call.

`long int nwd_write[2]` holds, on exit from a call to `ma77_analyse`, `ma77_factor`, `ma77_factor_solve`, and `ma77_solve`, the number of integer and real scalars written by HSL_OF01 during the subroutine call.

`int stat` holds the Fortran `stat` parameter.

`long int storage[4]` holds, on exit from `ma77_factor`, `ma77_factor_solve`, the maximum numbers of integers and reals that were stored in the superfiles `fname1`, `fname2`, `fname3`, `fname4` (or in the arrays that replaced the superfiles). In particular, `storage[2]` is the maximum multifrontal stack size.

int `tree_nodes` holds, on exit from `ma77_analyse`, the number of non-leaf nodes in the assembly tree (including any that are discarded but not reused).

int `unit_restart` holds, on exit from `ma77_finalise_restart` and `ma77_restart` the unit number of the sequential access files with name `restart_file`.

int `unused` holds, on exit from `ma77_analyse`, the number of indices in the range 1 to `n` that were not used for variables.

double `usmall` holds, on successful exit from `ma77_factor` or `ma77_factor_solve`, if `num_perturbed=0`, the final value of the relative pivot tolerance u and is set to zero otherwise.

2.6 Warning and error messages

A successful return from a subroutine in the package is indicated by `info.flag` having the value zero. A negative value is associated with an error message that by default will be output on 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 != 0` (true) and the matrix is found to be not positive definite. This error is also returned if `pos_def == 0` (false) and `control.action == 0` (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 `fname3`, or `fname4` already exists.
- 13 Returned by `ma77_open` or `ma77_restart` if `strlen(filename) > 400`. This error is also returned by `ma77_finalise` if `strlen(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 `strlen(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 `fname1` or `fname2` do not exist.
 - 29 Returned by `ma77_factor` and `ma77_factor_solve` if `pos_def! = 0` (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 == 0` (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! = 0` (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`.
- 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 != 0` (true) and the matrix is found to be singular.

2.7 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[]` and, if `ma77_factor` or `ma77_factor_solve` is called with `pos_def == 0` (false), in `control.storage_indef`. 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 `fname1` is 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 $i = 1, 2, \text{ or } 3$, 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 $i = 1 \text{ or } 2$, the superfile identified by `fname2` or `fname3` is used respectively.

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_finalize` is called with `restart_file` present, the matrix and factor integer and real data are written to superfiles identified by `fname1` and `fname2`. 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 2.5.19). 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: $n \geq 0$; $nvar \geq 0$; $strlen(path) \leq 400$; $strlen(fname1) \leq 400$; $strlen(fname2) \leq 400$; $strlen(fname3) \leq 400$; $strlen(fname4) \leq 400$; $strlen(restart_file) \leq 500$; $nrhs \geq 1$; $lx \geq n$; $lresid \geq n$; $1 \leq buffer_lpage[i] \leq file_size$, $i=0,1$ $buffer_npage[i] \geq 1$, $i=0,1$; $control.maxstore \geq 1$; $control.static = 0.0$ or $control.static \geq control.small$.

Portability: Fortran 2003 subset (F95 + TR15581 + C interoperability).

Changes from Version 5.6.0

Version 5.7.0 was the first to have a C interface.

Changes from Version 5

The alternative solve routine `MA77_solve_fredholm` and the multiply routine `MA77_multiply` have been added. The datatypes `ma77_control` and `ma77_info` have been expanded in size to allow for new entries in the future.

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 to 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-of-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 2×2 pivots. If a 2×2 pivot involving variables i and j is required, the user must set `order[i] < 0`, `order[j] < 0` and `|order[j]| = |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 $\bar{\mathbf{A}} = \mathbf{SAS}$ 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 non-zero (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 $v \geq \text{control.umin}$, the candidate is accepted and u is set to v . The new value is used in subsequent calls to `HSL_MA64`. If $v < \text{control.umin}$, 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 `fname4`. 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`.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "hsl_ma77d.h"

void error_exit(void**, struct ma77_control*, struct ma77_info*);

/* Simple code to illustrate row entry to hsl_ma77 */
int main(void) {
```

```
/* Derived types */
void *keep;
struct ma77_control control;
struct ma77_info info;

/* Parameters */
const int mvar = 10; /* largest number of entries in a row */

int i;

int *order;
double *x, *resid;

int index[mvar];
double values[mvar];
char* fname[4];

int lx, lresid, n, nrhs, nvar;
int idx;
int pos_def;

/* Read in the order n of the matrix */
scanf("%d\n", &n);
/* Choose file identifiers (hold direct access files in current directory)*/
fname[0] = "factor_integer";
fname[1] = "factor_real";
fname[2] = "work_real";
fname[3] = "templ";

ma77_default_control(&control);

/* Allocate arrays of appropriate size */
order = (int *) malloc(sizeof(int)*n);
x = (double *) malloc(sizeof(double)*n);
resid = (double *) malloc(sizeof(double)*n);

/* Initialisation */
ma77_open(n, fname[0], fname[1], fname[2], fname[3], &keep, &control,
         &info);

/* 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. */
for(idx=0; idx<n; idx++) {
    scanf("%d\n", &nvar);
    for(i=0; i<nvar; i++) scanf("%d", &index[i]);
    scanf("\n");
    for(i=0; i<nvar; i++) scanf("%lf", &values[i]);
    scanf("\n");
}
```

```
    ma77_input_vars(idx, nvar, index, &keep, &control, &info);
    if (info.flag < 0) error_exit(&keep, &control, &info);
    ma77_input_reals(idx, nvar, values, &keep, &control, &info);
    if (info.flag < 0) error_exit(&keep, &control, &info);
}

/* Use the natural pivot order 0,1,...,n-1 */
for(i=0; i<n; i++)
    order[i] = i;

/* Perform analyse and factorise */
ma77_analyse(order, &keep, &control, &info);
if (info.flag < 0) error_exit(&keep, &control, &info);
pos_def = 0; /* false */
ma77_factor(pos_def, &keep, &control, &info, NULL);
if (info.flag < 0) error_exit(&keep, &control, &info);

/* Read in the right-hand side and copy into resid. */
for(i=0; i<n; i++) scanf("%lf", &x[i]);
for(i=0; i<n; i++) resid[i] = x[i];
/* Solve and then compute the residuals */
nrhs = 1;
lx = n;
ma77_solve(0, nrhs, lx, x, &keep, &control, &info, NULL);
if (info.flag < 0) error_exit(&keep, &control, &info);
lresid = n;
ma77_resid(nrhs, lx, x, lresid, resid, &keep, &control, &info, NULL);
if (info.flag < 0) error_exit(&keep, &control, &info);
printf(" The computed solution is:\n");
for(i=0; i<n; i++) printf("%10.3lf ", x[i]);
printf("\n");
printf(" The residuals are:\n");
for(i=0; i<n; i++) printf("%10.3lf ", fabs(resid[i]));
printf("\n");

ma77_finalise(&keep, &control, &info);

/* Deallocate all arrays */
free(order);
free(x);
free(resid);

return 0;
}

/* Cleanup hsl_ma77 data structures then exit with an error */
void error_exit(void **keep, struct ma77_control *control,
    struct ma77_info *info) {
    ma77_finalise(keep, control, info);
```

```
    exit(1);
}
```

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 $\mathbf{A}^{(k)}$, $1 \leq k \leq 4$:

$$\begin{array}{cc}
 2 \begin{pmatrix} 2 & 1 \\ 1 & 7 \end{pmatrix} & 5 \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix} & 2 \begin{pmatrix} 4 & 3 & 2 & 3 \\ 3 & 10 & 3 & 2 \\ 2 & 3 & 6 & 1 \\ 3 & 2 & 1 & 6 \end{pmatrix} & 5 \begin{pmatrix} 4 & 1 & 1 & 3 \\ 1 & 3 & 2 & 2 \\ 1 & 2 & 2 & 1 \\ 3 & 2 & 1 & 4 \end{pmatrix}
 \end{array}$$

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 $1 \leq i \leq n$ ($n = 9$) 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`.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "hsl_ma77d.h"
```

```
void error_exit(void**, struct ma77_control*, struct ma77_info*);

/* Simple code to illustrate element entry to hsl_ma77 */
int main(void) {

    /* Derived types */
    void *keep;
    struct ma77_control control;
    struct ma77_info info;

    /* Parameters */
    const int mvar = 4; /* largest number of variables in an element */

    int i, irhs;

    int *order;
    double *x, *resid;

    int eltvar[mvar];
    double values[(mvar*mvar+mvar)/2];
    char* fname[4];
    int ielt, k1, lx, lresid, n, nelt, nrhs, nvar;
    int pos_def;

    /* Read in the order n, number of elements and number of right-hand sides */
    scanf("%d%d%d\n", &n, &nelt, &nrhs);

    /* Choose file identifiers (hold files in current directory)*/
    fname[0] = "factor_integer";
    fname[1] = "factor_real";
    fname[2] = "work_real";
    fname[3] = "templ";

    /* Allocate arrays of appropriate size */
    order = (int *) malloc(sizeof(int)*n);
    x = (double *) malloc(sizeof(double)*n*nrhs);
    resid = (double *) malloc(sizeof(double)*n*nrhs);

    /* Initialisation */
    ma77_default_control(&control);
    ma77_open_nelt(n, fname[0], fname[1], fname[2], fname[3], &keep, &control,
        &info, nelt);
    if(info.flag < 0) error_exit(&keep, &control, &info);

    /* For each element, read in the number of variables, the variable indices
     * and numerical values (lower triangular part of element). */
    for(ielt=0; ielt<nelt; ielt++) {
        scanf("%d\n", &nvar);
```

```

for(i=0; i<nvar; i++) scanf("%d", &eltvar[i]);
scanf("\n");
for(i=0; i<(nvar*nvar+nvar)/2; i++) scanf("%lf", &values[i]);
scanf("\n");
ma77_input_vars(ielt, nvar, eltvar, &keep, &control, &info);
if(info.flag < 0) error_exit(&keep, &control, &info);

/* 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. */
kl = 0;
for(i=0; i<nvar; i++) {
    ma77_input_reals(ielt, nvar-i, values+kl, &keep, &control, &info);
    if(info.flag < 0) error_exit(&keep, &control, &info);
    kl = kl + nvar-i;
}
}

/* Use the natural pivot order 0,1,...,n-1 */
for(i=0; i<n; i++)
    order[i] = i;

/* Perform analyse */
ma77_analyse(order, &keep, &control, &info);
if(info.flag < 0) error_exit(&keep, &control, &info);

/* Read in the right hand sides and copy into resid */
for(irhs=0; irhs<nrhs; irhs++) {
    for(i=0; i<n; i++) {
        scanf("%lf", &x[i+irhs*n]);
    }
}
for(i=0; i<n*nrhs; i++) resid[i] = x[i];

/* Perform factorisation and solve together */
pos_def = 1; /* true */
lx = n;
ma77_factor_solve(pos_def, &keep, &control, &info, NULL, nrhs, lx, x);
if(info.flag < 0) error_exit(&keep, &control, &info);

/* Compute the residuals */
lresid = n;
ma77_resid(nrhs, lx, x, lresid, resid, &keep, &control, &info, NULL);
if(info.flag < 0) error_exit(&keep, &control, &info);

for(irhs=0; irhs<nrhs; irhs++) {
    printf("\nSolution and residual for right-hand side %2i\n", irhs);
    for(i=0; i<n; i++){
        if(order[i] != 0){

```

```

        printf("%2i ", i);
        printf("%10.3lf ", x[i+irhs*lx]);
        printf("%10.3lf ", fabs(resid[i+irhs*lresid]));
        printf("\n");
    }
}

ma77_finalise(&keep, &control, &info);

/* Deallocate all arrays */
free(order);
free(x);
free(resid);

return 0;
}

/* Cleanup hsl_ma77 data structures then exit with an error */
void error_exit(void **keep, struct ma77_control *control,
                struct ma77_info *info) {
    ma77_finalise(keep, control, info);
    exit(1);
}

```

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. 3. -6. 0. -14. 10. 0. 8. 4.  
0. 3. -6. 0. -20. 8. 0. 4. -4.
```

This produces the following output:

Solution and residual for right-hand side 0

1	1.000	0.000
2	-1.000	0.000
3	0.000	0.000
4	-1.000	0.000
5	1.000	0.000
6	0.000	0.000
7	1.000	0.000
8	1.000	0.000

Solution and residual for right-hand side 1

1	1.000	0.000
2	-1.000	0.000
3	0.000	0.000
4	-1.000	0.000
5	1.000	0.000
6	0.000	0.000
7	1.000	0.000
8	-1.000	0.000