HSL_MA78

PACKAGE SPECIFICATION HSL

1 SUMMARY

HSL_MA78 solves one or more sets of sparse unsymmetric equations $\mathbf{AX} = \mathbf{B}$ or $\mathbf{A}^T\mathbf{X} = \mathbf{B}$ using an out-of-core multifrontal method. The $n \times n$ matrix $\mathbf{A}$ must be in unassembled element form, that is,

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

where 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. 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 2.3.5 and 2.3.7 (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 singular.

The multifrontal method is a variant of sparse Gaussian elimination. It involves the factorization

$$\mathbf{A} = \mathbf{PLDUQ}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are a permutation matrices, $\mathbf{L}$ and $\mathbf{U}$ are unit lower and upper triangular matrices, respectively, and $\mathbf{D}$ is a diagonal matrix. The factorization is performed by the subroutine `MA78_factor` and is controlled by an elimination tree that is constructed by the subroutine `MA78_analyse`, which needs the lists of variables in elements and an elimination sequence. Once a matrix has been factorized, any number of calls to the subroutine `MA78_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_MA78` is dependent on the elimination order that the user supplies. A suitable ordering may be obtained by first assembling the sparsity pattern of the matrix $\mathbf{A}$ (`MC57` can be used to do this) and then calling the HSL package `HSL_MC68`.

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 `MA78_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 2.3.14). 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. 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 `MA78_factor` and `MA78_solve` there are calls to the package `HSL_MA74` for the efficient partial factorization and partial solution of full sets of unsymmetric equations.

An option exists to scale the matrix. In this case, the factorization of the scaled matrix $\overline{\mathbf{A}} = \mathbf{S_r A S_c}$ is computed, where $\mathbf{S_r}$ and $\mathbf{S_c}$ are diagonal row and column scaling matrices.

ATTRIBUTES — Version: 3.6.0 (27th March 2023) **Types:** Real (single, double). **Uses:** `KB07`, `HSL_KB22`, `HSL_OF01`, `HSL_MA74`, and the `LAPACK` routine `_GETRF`. **Date:** March 2007; Version 3.3.0. October 2009. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Remark:** The development of HSL_MA78 was supported by EPSRC grants GR/S42170 and EP/E053351/1. For symmetric positive definite and symmetric indefinite systems, HSL_MA77 should be used.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement
Single precision version

```
USE HSL_MA78_single
```

Double precision version

```
USE HSL_MA78_double
```

If it is required to use more than one module at the same time, the derived types (Section 2.2) must be renamed in one of the USE statements.

The following procedures are available to the user:

- MA78_open must be called once for each problem to initialize the data structures and open the superfiles.
- MA78_input_vars must be called once for each element to specify which variables are associated with it.
- MA78_analyse must be called after all calls to MA78_input_vars are complete. The user must supply an elimination order that is used to construct the data structures needed for the factorization.
- MA78_input_reals must be called for each element 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 MA78_input_reals for a given element may be made at any time after the corresponding call to MA78_input_vars. All the reals must be input before MA78_factor is called. If the user enters data for an element that has previously been entered, the original data are discarded. If this is done after a call to MA78_factor, a new call to MA78_factor will be needed.
- MA78_scale may be called after all the reals of **A** have been input and after the call to MA78_analyse. If called, a scaling of the matrix is computed.
- MA78_factor may be called after all the reals of **A** have been input and after the call to MA78_analyse. If the user wishes to scale the matrix, MA78_scale must be called before the call to MA78_factor. The matrix **A** is factorized using the information from the call to MA78_analyse. Multiple calls to MA78_factor may follow a call to MA78_analyse.
- MA78_factor_solve may be called in place of MA78_factor to factorize **A** and, at the same time, solve the system $\mathbf{AX} = \mathbf{B}$. Multiple calls to MA78_factor_solve may follow a call to MA78_analyse.
- MA78_solve uses the computed factors generated by MA78_factor (or MA78_factor_solve) to solve systems $\mathbf{AX} = \mathbf{B}$ or $\mathbf{A}^\mathbf{T}\mathbf{X} = \mathbf{B}$. Multiple calls to MA78_solve may follow a call to MA78_factor. An option is available to perform a partial solution.
- MA78_resid may be called after a call to MA78_factor_solve or to MA78_solve to compute the residual matrix $\mathbf{B} - \mathbf{AX}$ (or $\mathbf{B} - \mathbf{A}^\mathbf{T}\mathbf{X}$).
- MA78_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.

- MA78_restart may be called after a call to MA78_finalise that filed the problem data. It restarts the computation and allows the user to solve for further right-hand sides or to factorize another matrix with the same structure.

2.2 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types MA78_control, MA78_info, and MA78_keep. The following pseudocode illustrates this.

```
use HSL_MA78_double
...
type (MA78_control) :: control
type (MA78_info) :: info
type (MA78_keep) :: keep
...
```

The components of MA78_control and MA78_info are explained in Sections 2.3.14 and 2.3.15, respectively. The components of MA78_keep are private and are used to pass data between the subroutines of the package.

2.3 Argument lists and calling sequences

2.3.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments. With the exception of the call to MA78_factor_solve, 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**.

2.3.2 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.

2.3.3 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.

2.3.4 The initialization subroutine

Data structures are set up and superfiles are opened by a call to MA78_open.

```
call MA78_open(n,nelt,filename,keep,control,info[,path])
```

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

nelt is a optional scalar INTENT(IN) argument of type INTEGER(short). It must be set to be at least the largest integer used to index an element. **Restriction:** nelt $\geq$ 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 superfiles that is used 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_MA78`, 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 work in-core (see Section 2.5). **Restriction:** `len(filename)` $\leq$ `400`.

`keep` is a scalar `INTENT(OUT)` argument of type `MA78_keep`. It is used to hold data about the problem being solved and must be passed unchanged to the other subroutines. `MA78_open` allocates its allocatable components.

`control` is a scalar `INTENT(IN)` argument of type `MA78_control`. Its components control the actions of the package, see Section 2.3.14.

`info` is a scalar `INTENT(OUT)` argument of type `MA78_info`. Its components provide information about the execution of the package, as explained in Section 2.3.15.

`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. 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`.

2.3.5 The input of integer data

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

```
call MA78_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. Each element may be input only once (it is **not** possible to change the variable list for an element without first calling `MA78_finalise` to terminate the computation, recalling `MA78_open` and then recalling `MA78_input_vars` for each element). If `index` is less than `1` or greater than `nelt`, the element is ignored.

`nvar` is a scalar `INTENT(IN)` argument of type `INTEGER(short)`. It must hold the number of variables in the incoming element. **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. Duplicates are allowed. Out-of-range indices are ignored.

`keep` is a scalar `INTENT(INOUT)` argument of type `MA78_keep` that must be passed unchanged by the user.

`control` is a scalar `INTENT(IN)` argument of type `MA78_control` (see Section 2.3.14).

`info` is a scalar `INTENT(INOUT)` argument of type `MA78_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.3.15. It must be passed unchanged by the user.

2.3.6 To analyse the sparsity pattern and prepare for the factorization

After completion of the sequence of calls to MA78_input_vars, a call of the following form must be made:

```
call MA78_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, order(i) must hold its position in the pivot sequence. 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 MA78_factor will be given; this order may give slightly more fill-in than the user-supplied order and may be modified by MA78_factor to maintain numerical stability.

keep, control, info: see Section 2.3.5.

2.3.7 The input of real data

The subroutine MA78_input_reals must be called for each element to specify its reals. The corresponding integer data must have already been entered. The data must be packed in the order given by the integer data into a full matrix held by columns with no gaps between columns. This is illustrated in Section 5. The reals for each element may be provided using a single call or, if the index lists that were passed to MA78_input_vars contained no duplicated or out-of-range indices, using a sequence of adjacent calls. Any previous real data for the element is discarded.

```
call MA78_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. If index is out-of-range, the element 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 $\geq$ 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 2.3.5.

2.3.8 To compute scaling factors

To compute a row and column scaling of the matrix **A**, a call of the following form may be made after the calls to MA78_input_reals are complete and after the call to MA78_analyse:

```
call MA78_scale(scale,keep,control,info[,anorm])
```

scale is a rank-1 array of size at least 2n of INTENT(OUT) and type REAL. On exit, scale(1:n) contains the diagonal entries of the row-scaling matrix $\mathbf{S}_r$ and scale(n+1:2n) contains the diagonal entries of the column-scaling matrix $\mathbf{S}_c$.

keep, control, info: see Section 2.3.5.

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).

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

To factorize the matrix, a call of the following form may be made after the calls to `MA78_input_reals` are complete and after the call to `MA78_analyse`:

```
call MA78_factor(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 MA78_factor_solve(keep,control,info,nrhs,lx,x[,scale)
```

`keep`, `control`, `info`: see Section 2.3.5.

`nrhs` is a scalar `INTENT(IN)` argument of type `INTEGER(short)` that holds the number of right-hand sides. **Restriction:** `nrhs` $\geq$ `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` $\geq$ `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 `2n` of `INTENT(IN)` and type `REAL`. If present, `scale(1:n)` must contain the diagonal entries of the row-scaling matrix $\mathbf{S}_r$ and `scale(n+1:2n)` must contain the diagonal entries of the column-scaling matrix $\mathbf{S}_c$.

2.3.10 To solve linear systems using the computed factors

After the call to `MA78_factor`, one or more calls of the following form may be made to solve $\mathbf{AX} = \mathbf{B}$ or $\mathbf{A^TX} = \mathbf{B}$. Partial solutions may be performed by appropriately setting the optional parameter `job`.

```
call MA78_solve(nrhs,lb,b,lx,x,keep,control,info[,scale,trans,job])
```

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

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

`b` is a rank-2 array with extents `lb` 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), `b(i,j)` holds the component of the right-hand side for variable `i` to the `j`th system. If `job` is absent, `b` is changed on exit; otherwise `b` in unchanged.

`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 `lx` and `nrhs` of `INTENT(OUT)` and type `REAL`. On exit, `x(i,j)` holds the solution for variable `i` to the `j`th system.

`keep`, `control`, `info`: see Section 2.3.5.

`scale` is an optional rank-1 array of size at least `2n` of `INTENT(IN)` and type `REAL`. If `scale` was present on the last call to `MA78_factor` (or `MA78_factor_solve`), it must also be present on each call to `MA78_solve` and `scale(1:2n)` must be unchanged since the call to `MA78_factor` (or `MA78_factor_solve`).

`trans` is an optional scalar `INTENT(IN)` argument of type `LOGICAL`. If present and set to `.true.`, transpose systems $\mathbf{A^T X = B}$ are solved; otherwise, systems $\mathbf{AX = B}$ are solved.

`job` is an optional scalar `INTENT(IN)` argument of type `INTEGER(short)`. If absent, $\mathbf{AX = B}$ or $\mathbf{A^T X = B}$ is solved. The factorization that has been computed may be expressed in the form

$$\mathbf{S_r A S_c = PLDUQ}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, $\mathbf{L}$ and $\mathbf{Q}$ are unit lower triangular and upper triangular matrices, respectively, and $\mathbf{D}$ is diagonal. $\mathbf{S_r}$ and $\mathbf{S_c}$ are diagonal scaling matrices and are equal to the identity unless `scale` was present on the call to `MA78_factor` (or `MA78_factor_solve`). A partial solution may be computed by setting `job` to have one of the following values:

`1` for solving $\mathbf{PLX = S_r B}$ (or $\mathbf{Q^T U^T X = S_r B}$ if `trans = .true`)

`2` for solving $\mathbf{DUQS_c^{-1} X = B}$ (or $\mathbf{DL^T P^T S_c^{-1} X = B}$ if `trans = .true`)

Restriction: `job = 1,2`.

2.3.11 To compute the residual matrix

Following a call to `MA78_factor_solve` or to `MA78_solve` (with `job` absent), the residual matrix $\mathbf{B - AX}$ or $\mathbf{B - A^T X}$ may be computed by making a call of the form:

```
call MA78_resid(nrhs,lx,x,lresid,resid,keep,control,info[,trans,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 $\mathbf{B - AX}$ (or $\mathbf{B - A^T X}$ if `trans = .true`).

`keep`, `control`, `info`: see Section 2.3.5.

`trans` is an optional scalar `INTENT(IN)` argument of type `LOGICAL`. If present and set to `.true.`, the residual $\mathbf{B - A^T X}$ is computed; otherwise, $\mathbf{B - AX}$ is computed.

`anorm_bnd` is an optional scalar `INTENT(OUT)` argument of type `REAL`. On exit, it holds $\sum_{k=1}^{m} ||\mathbf{A^{(k)}}||_\infty$, which is an upper bound for $||\mathbf{A}||_\infty$ or, if `trans` is present and set to `.true.`, it holds an upper bound for $||\mathbf{A^T}||_\infty$.

2.3.12 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 MA78_finalise(keep,control,info[,restart_file])
```

keep is a scalar INTENT(INOUT) argument of type MA78_keep that must be passed unchanged. On exit, allocatable components will have been deallocated.

control is a scalar INTENT(IN) argument of type MA78_control. Only the components that control printing are accessed (see Section 2.3.14).

info is a scalar INTENT(INOUT) argument of type MA78_info. Its components provide information about the execution of the subroutine, as explained in Section 2.3.15.

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_MA78 and are identified by filename(1:2) are saved and those identified by filename(3:4) (see Section 2.3.4) are deleted. **Restriction:** len(restart_file) $\leq$ 500.

2.3.13 The restart subroutine

If the user wishes to perform further factorizations or to solve for further right-hand sides after a call to MA78_finalise, a call of the following form should be made:

```
call MA78_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 MA78_finalise. The file must not have been changed since then. On successful exit, this file will be closed and deleted. **Restriction:** len(restart_file) $\leq$ 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 MA78_finalise. Files with names path(i)//filename(j) must not exist for j = 3, 4 and any value of i in the range 1, 2, ..., SIZE(path). **Restriction:** len(filename) $\leq$ 400.

keep is a scalar INTENT(OUT) argument of type MA78_keep. On exit, its allocatable components that are required by MA78_solve or MA78_factor will have been allocated and the contents restored.

control is a scalar INTENT(IN) argument of type MA78_control. Only the components that control printing are accessed (see Section 2.3.14).

info is a scalar INTENT(INOUT) argument of type MA78_info. Its components provide information about the execution of the subroutine, as explained in Section 2.3.15.

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) $\leq$ 400.

2.3.14 The derived data type for holding control parameters

The derived data type `MA78_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 `MA78_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` $\leq$ `buffer_lpage(:)` $\leq$ `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(:)` $\geq$ `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 2.5. The default is `maxstore=0`. **Restriction:** `maxstore` $\geq$ `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 4 and type INTEGER(short). If the user wants to use arrays in place of the superfiles filename(1:4), storage(1:4) 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 = 1:4). Further details are given in Section 2.5. The default is storage(:)=0.

Controls used by MA78_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=16. The default is used if nemin < 1.

Controls used by MA78_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 $1 \pm$ thresh.

Controls used by MA78_factor

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.

lfactor is a scalar of type default LOGICAL. If lfactor = .true. on entry to MA78_factor_solve, the **PL** factor is stored. If the user wishes to minimise the amount of i/o by not storing the **PL** factor, lfactor should be set to .false. The **PL** factor must be stored if the user wishes to call MA78_solve to solve either for further right-hand sides or transpose systems. The default is lfactor = .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. The default value is 1.1.

nb is a scalar of type INTEGER(short) that holds the block size used within the kernel factorization code HSL_MA74. This is discussed further in Section 2.2.6 of the HSL_MA74 specification. The default is nb=80. The default is used if nb < 1.

pivoting is a scalar of type INTEGER with default value 1 that controls the pivoting. Possible values are:

1 : threshold partial pivoting.

2 : threshold diagonal pivoting.

3 : threshold rook pivoting.

Restriction: pivoting = 1, 2 or 3.

small is a scalar of type REAL with default value 1×10^{-20}. If, during the factorization, all the entries in a column of the reduced matrix are of modulus less than or equal to small, all the entries in the column are replaced by zero. Every pivot (nonzero entry of *D*) must also be of absolute value greater than the absolute value of small.

static is a scalar of type REAL with default value 0.0. If static is positive, pivots that do not satisfy the threshold criteria may be selected and small pivots may be replaced by static until all the eliminations are completed; in this case, the factorization may be inaccurate. See Section 4 for details. **Restriction:** static = small or static $\geq$ abs(small).

u is a scalar of type REAL and default value 0.01 that holds the pivoting threshold parameter. Values of u that are less than 0.0 are treated as 0.0 and values greater than 1.0 are treated as 1.0. Values near 0.0 may give a faster factorization with fewer entries in the factors but may result in a less stable factorization.

2.3.15 The derived data type for holding information

The derived data type MA78_info is used to hold parameters that give information about the progress and needs of the algorithm. The components of MA78_info (in alphabetical order) are:

detlog is a scalar of type REAL. On exit from MA78_factor and MA78_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 MA78_factor and MA78_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 2.4).

index is an array of length 4 and type INTEGER(short). On exit from MA78_open and MA78_restart, index(i) holds the HSL_OF01 index of the superfile identified by filename(i) (i = 1,2,3,4). On exit from MA78_factor and MA78_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 MA78_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 MA78_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 MA78_factor and MA78_factor_solve, it holds the computed rank of the factorized matrix.

maxdepth is a scalar of type INTEGER(short). On exit from MA78_analyse, it holds the maximum depth of the assembly tree.

maxfront is a scalar of type INTEGER(short). On exit from MA78_analyse, it holds the maximum front size (assuming the pivot sequence can be used without modification). On exit from MA78_factor and MA78_factor_solve, it holds the maximum front size.

minstore is a scalar of type INTEGER(long). On exit from MA78_factor and MA78_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 `MA78_factor` and `MA78_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 `MA78_analyse`, it holds the number of entries that will be in the **L** or **U** factor (assuming the pivot sequence can be used without modification). On exit from `MA78_factor` and `MA78_factor_solve`, it holds the actual number of entries in the **L** or **U** factor.

`nflops` is scalar of type `INTEGER(long)`. On exit from `MA78_analyse`, it holds the number of floating-point operations that will be needed to perform the factorization (assuming the pivot sequence can be used without modification). On exit from `MA78_factor` and `MA78_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 `MA78_analyse`, `MA78_factor`, `MA78_factor_solve`, and `MA78_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 `MA78_analyse`, `MA78_factor`, `MA78_factor_solve`, and `MA78_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 `MA78_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 `MA78_input_vars`, it holds the number of supervariables in the problem (see Section 4).

`num_diag` is a scalar of type `INTEGER`. On successful exit, it holds the number of pivots that were chosen from the diagonal.

`num_file` is an array of size 4 of type `INTEGER(short)`. On exit from a call to `MA78_finalise`, `num_file(1:4)` holds the number of secondary files used by each of the superfiles.

`num_nothresh` is a scalar of type `INTEGER`. On successful exit from `MA78_factor`, it holds the number of pivots which did not satisfy the threshold criteria based on the value of `control%u`.

`num_perturbed` is a scalar of type `INTEGER`. On successful exit from `MA78_factor`, it holds number of pivots that were replaced by `control%static`.

`nwd_read` is an array of size 2 of type `INTEGER(long)`. On exit from a call to `MA78_analyse`, `MA78_factor`, `MA78_factor_solve`, and `MA78_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 `MA78_analyse`, `MA78_factor`, `MA78_factor_solve`, and `MA78_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 `MA78_factor` and `MA78_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).

`tree_nodes` is a scalar of type `INTEGER(short)`. On exit from `MA78_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 `MA78_finalise` and `MA78_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 `MA78_analyse`, it holds the number of indices in the range `1` to `n` that were not used for variables.

`usmall` is a scalar of type `REAL`. On successful exit from `MA78_factor`, if `num_perturbed = 0`, `usmall` holds the threshold parameter that was used and is set to zero otherwise.

2.4 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 `MA78_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 `MA78_input_vars` if a call has already been made for the current element. Immediate return.

-10 Returned by `MA78_input_reals` if `MA78_input_vars` has not been called for the current element. Immediate return.

-11 Returned by `MA78_factor` and `MA78_factor_solve` if `control%action = .false.` and the matrix is found to be singular.

-12 Returned by `MA78_open` if a file of the given name already exists. This error is also returned by `MA78_finalise` if a file of name `restart_file` already exists and by `MA78_restart` if a file identified by `filename(3:4)` already exists.

-13 Returned by `MA78_open` or `MA78_restart` if `len(filename)` $>$ `400`. This error is also returned by `MA78_finalise` if `len(restart_file)` $>$ `500`.

-14 Returned by `MA78_input_reals` if the data for the previous element 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 MA78_input_reals if there are duplicated or out-of-range entries in one or more of the element variable lists and user has not entered all the reals for the current element in a single call to MA78_input_reals.

−18 Returned by MA78_open if $\texttt{nelt} < 0$.

−19 Returned by MA78_input_reals if $\texttt{length} < 0$. Immediate return.

−20 Returned by MA78_solve if job is out of range.

−21 Returned by MA78_analyse if an error is found in the user-supplied elimination order (held in order). Immediate return.

−22 Returned by MA78_factor, MA78_factor_solve and MA78_scale if for one or more of the elements, MA78_input_vars was called but no corresponding call was made to MA78_input_reals.

−23 Returned by MA78_open if control%buffer_lpage(:)<1 or control%buffer_lpage(:)>control%file_size. Immediate return.

−24 Returned by MA78_factor_solve, MA78_solve, and MA78_resid if there is an error in the size of array x (that is, $\texttt{lx} < \texttt{n}$ or $\texttt{nrhs} < 1$).

−25 Returned by MA78_solve if $\texttt{lb} < \texttt{n}$.

−26 Returned by MA78_open if control%buffer_npage(:) < 1. Immediate return.

−27 Returned by MA78_open if control%maxstore is out of range. Immediate return.

−28 Returned by MA78_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 MA78_factor_solve if there is insufficient memory to allocate a two-dimensional workarray with first dimension n and second dimension equal to the number of right-hand sides. In the event of this error, the user should call MA78_factor and then MA78_solve once the factorization is complete.

−30 Returned by MA78_factor, MA78_factor_solve and MA78_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 MA78_input_vars if all the variable indices in an element are out-of-range.

−32 Returned by MA78_input_reals if more than the expected number of reals have been entered for the current element. Immediate return.

−33 Returned by MA78_input_vars if $\texttt{nvar} < 0$. Immediate return.

−34 Returned by MA78_resid if $\texttt{lresid} < \texttt{n}$.

−35 Immediate return from MA78_factor and MA78_factor_solve if control%pivoting is out of range.

−36 Immediate return from MA78_factor and MA78_factor_solve if $\texttt{control\%static} < \texttt{abs}(\texttt{control\%small})$ and $\texttt{control\%static} \neq 0.0$.

−37 Returned by MA78_solve if call follows a call to MA78_factor_solve with $\texttt{control\%lfactor} = \texttt{.false.}$ (**PL** factor was not stored).

−38 Returned by MA78_scale, MA78_factor, MA78_factor_solve, and MA78_solve if the size of the array scale is too small. This error is also returned by MA78_solve if scale is absent when it was present on the call to MA78_factor (or MA78_factor_solve), or if scale is present when it was not present on the call to MA78_factor.

−39 Returned by MA78_factor IEEE infinities found in the reduced matrix, probably caused by control%small or control%u having too small a value.

−41 Returned by MA78_finalise if there is an error in a Fortran CLOSE statement.

A positive value of info%flag on exit from MA78_factor and MA78_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 MA78_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 MA78_input_vars if duplicated variable indices have been found in the user-supplied array list. Duplicates are recorded and the corresponding reals are by MA78_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 MA78_input_vars if both out-of-range and duplicated variable indices have been found in the user-supplied array list.

+4 Returned by MA78_factor and MA78_factor_solve if control%action = .true. and the matrix is found to be singular.

+5 Returned by MA78_factor and MA78_factor_solve if control%pivoting = 2 and some off diagonal pivots were chosen.

+6 Returned by MA78_factor and MA78_factor_solve if both warnings 4 and 5 have been issued.

2.5 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(1:4). If control%maxstore >0 and the user does not set control%storage, 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 superfile identified by filename(1) 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 = 1:4, 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).

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 and control%storage for subsequent runs.

If MA78_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 MA78_restart.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: KB07, HSL_KB22, HSL_OF01, HSL_MA74, and the Lapack routine _GETRF.

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 2.3.14). 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_MA78.

Restrictions: n≥0; nvar≥0; len(path)≤400; len(filename)≤400; len(restart_file)≤500; control%pivoting = 1, 2, or 3; control%static = 0.0 or control%static≥abs(control%small); nrhs≥1; lx≥n; lresid≥n.

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

Changes from Version 1
Version 2 includes an option to scale the matrix. 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.

4 METHOD

MA78_open

MA78_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.

MA78_input_vars

MA78_input_vars must be called for each 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 MA78_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 element contained any duplicated or out-or-range indices, a mapping array of length nvar that records the position of each variable in the 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 MA78_input_vars.

Note that, having input the variable list for a particular element, the user may not input data for this element again without first calling MA78_finalize and then recalling MA78_open followed by MA78_input_vars for each element.

MA78_analyse

MA78_analyse must be called once after all the calls to MA78_input_vars are complete (it may be called before or after the calls to MA78_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 MC57 should be called first to assemble the sparsity pattern of **A**.

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`.

MA78_input_reals

For each row or element, `MA78_input_reals` must be called, after the corresponding call to `MA78_input_vars` and before a call to `MA78_factor` or `MA78_factor_solve`. If the call to `MA78_input_vars` found duplicated or out-of-range indices, all the reals for that element must be input on a single call to `MA78_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 element and that the number of reals does not exceed the expected number. If real data has already been supplied for the incoming element, it is overwritten by the new data (this allows the reals of a matrix to change without the using having to recall `MA78_input_vars` and `MA78_analyse`).

If duplicated or out-of-range indices were input on the call to `MA78_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`.

MA78_scale

`MA78_scale` computes the scaling diagonal matrix $\mathbf{S}$ 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 **MA78_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_MA78` are called. Details of the scaling algorithm and how it is implemented within `HSL_MA78` are given in [3]. `MA78_scale` includes an option to compute the infinity norm of the matrix $\mathbf{A}$.

MA78_factor

`MA78_factor` performs the numerical factorization using the assembly tree and the ordering of the children that was set up by `MA78_analyse`. If scaling factors are input, the factorization of the scaled matrix $\overline{\mathbf{A}} = \mathbf{S_r A S_c}$ is computed. The nodes of the tree are visited in depth first search order. At each node, the partial factorization of the frontal matrix is performed by `HSL_MA74` (at the root node, if threshold partial pivoting is being used, that is, `control%pivoting = 1`, the Lapack routine `_GETRF` is used). If a pivot candidate does not satisfy the threshold pivot condition, either it is delayed or, if `control%static` is positive, pivots that come closest to satisfying this condition are chosen. In this case, the factorization may be inaccurate. 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 wishes to solve $\mathbf{AX} = \mathbf{B}$ at the same time as factorizing the matrix, the call to `MA78_factor` should be replaced by a call to `MA78_factor_solve`. The user must pass right-hand vectors to `MA78_factor_solve` using the argument `x`. The forward substitutions are performed as the factor entries are generated. Once the factorization is complete, `MA78_factor_solve` performs the back substitutions by calling `MA78_solve` with `job = 2`. The advantage of using `MA78_factor_solve` in place of `MA78_factor` followed by `MA78_solve` is that the former only requires the

DUQ factor to be read back in but the latter requires both the **PL** and **DUQ** factors to be read back in.

MA78_solve
Having checked the user's data, `MA78_solve` performs forward substitution followed by back substitution. At each of the tree, the partial forward and back substitutions are performed by `MA74_solve`. The matrix factors **PL** and **DUQ** must be accessed once, independently of the number of right-hand sides. Thus solving for several right-hand sides at once is significantly faster than repeatedly solving for a single right-hand side.

Reference:

J.K Reid and J.A. Scott. (2007). An efficient out-of-core multifrontal solver for element problems. RAL Technical Report. RAL-TR-2007-014.

[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.

5 EXAMPLE OF USE

We wish to solve a problem comprising the following four elemental matrices $\mathbf{A}^{(\mathbf{k})}$, $1 \le k \le 4$:

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

where the variable indices are indicated by the integers before each matrix. This matrix is used to solve a linear system with right hand sides,

$$\mathbf{B} = \begin{pmatrix} 12 & 28 & 14 & 15 & 30 & 20 \\ 31 & 104 & 49 & 52 & 107 & 101 \end{pmatrix}^T$$

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 `MA78_input_reals` is used to enter the real data. The solve is performed at the same time as the factorization by calling `MA78_factor` with the two right-hand sides present.

```
program example
! Simple code to illustrate hsl_ma78
      use hsl_ma78_double
      implicit none

! Derived types
      type (ma78_keep)    :: keep
      type (ma78_control) :: control
      type (ma78_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)
      character(len=20) :: path(1),filename(4)
      integer :: i,ielt,irhs,k,lx,lresid,n,nelt,nrhs,nvar

! 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)
      path(1) = ''
      filename(1) = 'factor_integer'
      filename(2) = 'factor_real'
      filename(3) = 'work_real'
      filename(4) = 'temp1'

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

! Initialisation
      call ma78_open(n,nelt,filename,keep,control,info)
      if (info%flag < 0) go to 100

! For each element, read in the number of variables, the
! variable indices and numerical values.
      do ielt = 1,nelt
        read (*,*) nvar
        read (*,*) eltvar(1:nvar)
        read (*,*) values(1:nvar*nvar)
        call ma78_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 ma78_input_reals,
! we enter the reals of the element matrix one column at a time.
        k = 1
        do i = 1,nvar
          call ma78_input_reals(ielt,nvar,values(k:k+nvar-1),keep, &
               control,info)
          if (info%flag < 0) go to 100
          k = k + nvar
        end do
      end do

! Use the natural pivot order 1,2,...,n
      do i = 1,n
```

```
        order(i) = i
      end do

! Perform analyse
      call ma78_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
      call ma78_factor_solve(keep,control,info,nrhs,lx,x)
      if (info%flag < 0) go to 100

! Compute the residuals
      call ma78_resid(nrhs,lx,x,lresid,resid,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:',abs(resid(1:n,irhs))
      end do

 100  call ma78_finalise(keep,control,info)

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

The input data used for this problem is:

```
 6 4 2
 2
 4 5
 2.0 -1.0 1.0 7.0
 2
 5 6
 3.0 4.0 2.0 8.0
 4
 4 5 1 2
 4.0 -1.0 2.0 3.0 3.0 1.0 3.0 2.0 2.0 3.0 6.0 1.0 3.0 2.0 1.0 5.0
 4
 5 6 2 3
 2.0 1.0 8.0 3.0 1.0 3.0 2.0 2.0 8.0 2.0 2.0 5.0 3.0 2.0 5.0 4.0
 12.0 28.0 14.0 15.0 30.0 20.0
```

```
 31.0 104.0 49.0 52.0 107.0 101.0
```

This produces the following output:

```
 For right-hand side  1

 The computed solution is:
     1.000     1.000     1.000     1.000     1.000     1.000

 The residuals are:
     0.000     0.000     0.000     0.000     0.000     0.000

 For right-hand side  2

 The computed solution is:
     1.000     2.000     3.000     4.000     5.000     6.000

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