Science and Technology Facilities Council

HSL_MA42_ELEMENT

PACKAGE SPECIFICATION HSL

1 SUMMARY

HSL_MA42_ELEMENT **solves one or more sets of sparse linear unsymmetric unassembled finite-element equations, $\mathbf{AX}=\mathbf{B}$, or $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$, or $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$, by the frontal method,** optionally using direct access files for the matrix factors. The system may be real or complex ($\mathbf{A}^{\mathrm{H}}$ denotes the conjugate transpose of $\mathbf{A}$). There are options for automatically ordering the elements, for supplying the elements using a reverse communication interface, for holding the matrix factors in direct-access files, and for preserving a partial factorization.

The $n\times n$ coefficient matrix $\mathbf{A}$ must have a symmetric structure and must be in elemental form

$$\mathbf{A}=\sum_{k=1}^{nelt}\mathbf{A}^{[k]},$$

where $\mathbf{A}^{[k]}$ is nonzero only in those rows and columns that correspond to variables in the k-th finite element. The elements must be square elements, with the row indices equal to the column indices. For each k, the user must supply a list specifying which rows/columns of $\mathbf{A}$ are associated with $\mathbf{A}^{[k]}$, and an array containing the nonzero entries. The right-hand sides $\mathbf{B}$ may be supplied in elemental form (that is, $\mathbf{B}=\sum_{k=1}^{nelt}\mathbf{B}^{[k]}$) or in assembled form.

The frontal method is a variant of Gaussian elimination and involves the factorization

$$\mathbf{A}=\mathbf{PLUQ},$$

where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, $\mathbf{L}$ is a unit lower triangular matrix, and $\mathbf{U}$ is an upper triangular matrix. The matrices $\mathbf{A}^{[k]}$ are assembled one at a time into a frontal matrix. At an intermediate stage of the factorization, the 'front' contains those variables associated with one or more of the $\mathbf{A}^{[k]}$ that have already been assembled and are also present in one or more of the $\mathbf{A}^{[k]}$ that have still to be assembled. The frontal matrix is held in main memory but the factors may be held in direct-access files. During the factorization, data is put into buffers (workspace arrays) then, once a buffer is full, its contents are written to a direct-access file. The user is able to control the amount of main memory required by choosing the sizes of the buffers. Three buffers and files may be used: one for the integer data, and one each for the entries in the $\mathbf{U}$ and $\mathbf{L}$ factors. Storage requirements may be reduced further by choosing not to store $\mathbf{L}$; $\mathbf{L}$ only needs to be stored if the user wishes either to solve subsequently for further right-hand sides or to solve transpose systems $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ or, in the complex case, conjugate systems $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$.

Because the efficiency of the frontal method (in terms of flop counts and memory requirements) is dependent on ordering the elements to keep the frontal matrix small, an option is included for automatically ordering the elements.

ATTRIBUTES — Version: 3.0.0. 27 April 2022 **Types:** Real (double), Complex (double). **Precision:** Double – Note: only double precision versions are provided, as **Remark:** For assembled systems of equations, HSL_MA42 may be used. **Calls:** MC63, HSL_MC73, I_AMAX, _AXPY, _GER, _GERU, _GEMV, _TPSV, _GEMM, _TRSM. **Language:** Fortran 95 + TR15581 (allocatable components). **Original date:** June 2004. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement

Double precision version

```
USE HSL_MA42_ELEMENT_double
```

Double complex version

```
USE HSL_MA42_ELEMENT_double_complex
```

If it is required to use two modules at the same time, the derived types (Section 2.1.4) must be renamed in one of the USE statements.

2.1.1 Calling sequence for reverse communication interface

If the user wishes to enter the matrix data element-by-element, the following subroutines must be called for each problem:

MA42_ELEMENT_START: must be called once to initialize the derived types.

MA42_ELEMENT_ANALYSE: must be called for each element to specify which variables are associated with it. An element assembly order is optionally generated. This can add a significant overhead (in terms of time and integer storage) but can substantially reduce the overall time and memory needed by the factorization and solve phases.

MA42_ELEMENT_FACTORIZE: must be called for each element to specify the entries of $\mathbf{A}^{[k]}$ and, optionally, $\mathbf{B}^{[k]}$. The calls must be made in the order determined by MA42_ELEMENT_ANALYSE. Data from MA42_ELEMENT_ANALYSE is used to factorize the matrix and, if element right-hand side matrices $\mathbf{B}^{[k]}$ are specified, the equations $\mathbf{AX}=\mathbf{B}$ with right-hand side(s) $\mathbf{B}=\sum\mathbf{B}^{[k]}$ are solved after the call for the last element. Note that more than one finite-element problem having elements with the same variable lists (but different numerical values) may be factorized and solved following a single set of calls to MA42_ELEMENT_ANALYSE.

In addition, the following subroutines may be called.

MA42_ELEMENT_PARTIAL: If the user chooses not to complete the sequence of calls to MA42_ELEMENT_FACTORIZE, MA42_ELEMENT_PARTIAL may be called to restore internal arrays and variables to the values they held prior to the first call to MA42_ELEMENT_FACTORIZE. This allows the factorization to be restarted without repeating the calls to MA42_ELEMENT_ANALYSE. MA42_ELEMENT_PARTIAL also provides the user with access to the data remaining in the frontal matrix at the point at which the factorization was terminated (see Section 2.5).

MA42_ELEMENT_SOLVE: uses the computed factors to rapidly solve either further systems of the form $\mathbf{AX}=\mathbf{B}$ or systems of the form $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ or $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$, with the right-hand side vectors $\mathbf{B}$ input in assembled form. An option exists to perform only forward elimination or back substitution.

MA42_ELEMENT_RESIDUAL: may be called for each element to compute the residuals $\mathbf{R}=\mathbf{B}-\mathbf{AX}$ (or $\mathbf{R}=\mathbf{B}-\mathbf{A}^{\mathrm{T}}\mathbf{X}$ or $\mathbf{R}=\mathbf{B}-\mathbf{A}^{\mathrm{H}}\mathbf{X}$).

Once all other calls to subroutines in the module are complete for a particular problem, a call must be made to:

MA42_ELEMENT_FINAL: deallocates all arrays that have been allocated by the module for the problem and deletes files used by the module for the problem with status SCRATCH.

2.1.2 Calling sequence avoiding reverse communication

If the user prefers not to use reverse communication, MA42_ELEMENT_START must first be called to initialize the derived types and then MA42_ELEMENT_AFS may be called to perform the analyse, factorize, and (optionally) solve phases in a single step. The user must supply all the element data in arrays or files. MA42_ELEMENT_SOLVE may be called to solve for further right-hand sides or to solve transposed or conjugate systems. MA42_ELEMENT_FINAL

should be called once after all other calls to subroutines in the module are complete for the problem.

Note that using MA42_ELEMENT_AFS does not offer all the options of the reverse communication interface. Unless the problem is small, the factors are held in direct-access files with status SCRATCH.

2.1.3 Package types

We use the term **package type** to mean double precision real for the double precision version, and double precision complex for the double complex version. That is,

REAL(kind(0.0d0)) in HSL_MA42_ELEMENT_double,

COMPLEX(kind(0.0d0)) in HSL_MA42_ELEMENT_double_complex.

We also use **real type** to mean double precision real for both the double precision and double precision complex versions. That is,

REAL(kind(0.0d0)) in HSL_MA42_ELEMENT_double and HSL_MA42_ELEMENT_double_complex.

In the following, INTEGER(long) denotes INTEGER(kind = selected_int_kind(18)) and INTEGER denotes default integers. All logicals are default logicals, and all characters are default characters.

2.1.4 The derived types

For each problem, the user must employ derived types defined by the module to declare structures of the following types: MA42_ELEMENT_CONTROL, MA42_ELEMENT_INFO, MA42_ELEMENT_KEEP, and MA42_ELEMENT. The following pseudocode for using the double precision version illustrates this.

```
USE HSL_MA42_ELEMENT_double
...
TYPE (MA42_ELEMENT_CONTROL) control
TYPE (MA42_ELEMENT_INFO) info
TYPE (MA42_ELEMENT_KEEP) keep
TYPE (MA42_ELEMENT) element
...
```

The components of the derived types control and info are explained in Sections 2.3 and 2.4, respectively. The components of keep are used to pass data between the subroutines within the package and should not be changed by the user at any stage of the computation; the components that may be of interest to the user are explained in Section 2.5. The components of element are:

mvar is an INTEGER scalar that is used to hold the largest number of variables in a single element.

nvar is an INTEGER scalar that is used to hold the number of variables in an element.

reals is a rank-2 pointer array of package type that is used to hold the numerical values of the entries of an element matrix.

rhs is a rank-2 pointer array of package type that is used to hold element right-hand side matrices.

vars is a rank-1 INTEGER pointer array that is used to hold the row/column indices of the variables in an element.

We remark that many components of the derived types are pointers but we use these pointers as if they were allocatable arrays. The Fortran 95 standard does not permit components of derived datatypes to be allocatable arrays; when the Fortran standard does permit this, we may replace all the pointer components used by HSL_MA42_ELEMENT by allocatable arrays.

2.2 Argument lists

We use square brackets [] to indicate optional arguments. In each call, optional arguments follow the argument control. 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.**

2.2.1 The initialization subroutine

For each problem, the user must first call the initialization subroutine MA42_ELEMENT_START.

```
CALL MA42_ELEMENT_START(element,keep,control)
```

element is a scalar structure of INTENT(OUT) and of type MA42_ELEMENT. On exit, its components that are pointer arrays will have been nullified.

keep is a scalar structure of INTENT(OUT) and of type MA42_ELEMENT_KEEP. On exit, its components that are pointer arrays will have been nullified. At no stage of the computation should keep be changed by the user.

control is a scalar structure of INTENT(OUT) and of type MA42_ELEMENT_CONTROL. On exit, its components will have been given the default values specified in Section 2.3. If the user wishes to use values other than the defaults, the corresponding components of control should be reset after the call to MA42_ELEMENT_START.

2.2.2 To perform the analysis and optionally order the elements

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

```
CALL MA42_ELEMENT_ANALYSE(n,nelt,element,order,keep,info,control)
```

n is an INTEGER scalar of INTENT(IN) that must be set to be at least as large as the largest integer used to index a variable. This argument is only accessed on the first call. **Restriction:** $n \geq 1$.

nelt is an INTEGER scalar of INTENT(IN) that must be set to the total number of finite elements (that is, to the total number of calls to be made to this routine). **Restriction:** $nelt \geq 1$.

element is a scalar of derived type MA42_ELEMENT and INTENT(IN). On the *k*-th call, the following components must be set by the user to hold data for the element matrix $\mathbf{A}^{[k]}$:

element%nvar is an INTEGER scalar that must hold the number of variables in element k. **Restriction:** $element\%nvar \geq 1$.

element%vars is a rank-1 INTEGER pointer array that must be allocated with size at least element%nvar and set to hold the indices of the variables in element k. These indices need not be in increasing order but must be distinct. **Restrictions:** $1 \leq element\%vars(:) \leq n$ and $element\%vars(I) \neq element\%vars(J)$, $I \neq J$.

order is a rank-1 INTEGER array of INTENT(INOUT) and size nelt. order must be set on entry if $control\%order63 = 0$ (the default value is 6). In this case, order must hold the element assembly order. For other values of control%order63, a suitable element assembly order is computed by the package (see control%order63 in Section 2.3 for more details). On exit, order holds the order in which the elements must be passed to MA42_ELEMENT_FACTORIZE.

keep is a scalar of derived type MA42_ELEMENT_KEEP and INTENT(INOUT) that must not be changed by the user between calls.

info is a scalar of derived type MA42_ELEMENT_INFO and INTENT(INOUT). Its components provide information about the execution of the subroutine, as explained in Section 2.4.

control is a scalar of derived type MA42_ELEMENT_CONTROL of INTENT(IN). Its components control the action, as explained in Section 2.3.

2.2.3 To factorize A and optionally solve $\mathbf{AX} = \mathbf{B}$

A call of the following form must be made for each element. **The elements must be passed in the order** order(1), order(2) , ..., order(nelt), **where** order **is as on exit from the final call to** MA42_ELEMENT_ANALYSE.

```
CALL MA42_ELEMENT_FACTORIZE(nrhs,element,keep,info,control  &
                            [,lenbuf,frontsize,filnam,x])
```

nrhs is an INTEGER scalar of INTENT(IN) that must be set to the number of right-hand sides that the user wants to enter in elemental form (that is, the number of linear systems $\mathbf{AX}=\mathbf{B}$ to be solved with $\mathbf{B}=\sum\mathbf{B}^{[k]}$). It is only accessed on the first call. A negative value is treated as zero.

element is a scalar of derived type MA42_ELEMENT and INTENT(INOUT). On the *k*-th call, the components element%nvar and element%vars must be set as in the call to MA42_ELEMENT_ANALYSE for element kelt=order(k) (note that no checks are made in this call for duplicate entries). In addition, the following components must be set:

element%mvar is an INTEGER scalar that must be at least as large as the largest number of variables in an element (see info%mvar in Section 2.4).

element%reals is a rank-2 pointer array of package type of size element%mvar by at least element%nvar that must contain the numerical values of the entries of the element matrix $\mathbf{A}^{[\mathtt{kelt}]}$ in packed form. That is, element%reals(I,J) must contain the contribution from element kelt to entry element%vars(I), element%vars(J) in the matrix **A** (I, J = 1, 2,..., element%nvar). This component is changed by the subroutine.

If nrhs > 0, the following component must also be allocated and set:

element%rhs is a rank-2 pointer array of package type of size element%mvar by at least nrhs that must contain $\mathbf{B}^{[\mathtt{kelt}]}$ in packed form. That is, element%rhs(I,J) must contain the contribution from element kelt to component element%vars(I) of the J-th right-hand side (I = 1, 2,..., element%nvar, J = 1, 2,..., nrhs). Contributions to the same component from different elements are summed. This component is changed by the subroutine.

keep, info, control are as in the calls to MA42_ELEMENT_ANALYSE.

lenbuf is an optional INTEGER array of INTENT(IN) and size 3. If present, lenbuf is only accessed on the first call and must hold the sizes (in words) of the buffers (work arrays) used for the **U** and **L** factors, and the integer factor data. If lenbuf is not present, buffer sizes will be chosen by the code. Further details are given in Section 2.8. **Restriction:** lenbuf(1) ≥ 1, lenbuf(3) ≥ 1 and, if control%lfactor = .true., lenbuf(2) ≥ 1.

frontsize is an optional scalar of type INTEGER of INTENT(IN). If present, frontsize is only accessed on the first call and must be set to the maximum frontsize. If frontsize is not present, a suitable value for the maximum frontsize is chosen by the code (see Section 2.7). **Restriction:** frontsize ≥ info%front_bound.

filnam is an optional rank-1 array of size 3 of type CHARACTER and length at most 128 and of INTENT(IN). If the user wishes to hold the factors in direct-access files, filnam should be present on the first call and must hold the names of the direct-access files for **U**, **L**, and the integer factor data, respectively. The names must all be different. If filnam is not present and direct-access files are needed because the buffers are not large enough to hold the factors, files with status SCRATCH are used.

x is an optional rank-2 array of package type and of INTENT(OUT) and size at least info%lindex by nrhs that is used to hold the solution. x is only accessed on the final call if nrhs > 0 on the first call. In this case, on exit from the final call, if I has been used to index a variable, x(I,J) holds the solution for variable I to system J (I = 1,2,..., info%lindex, J = 1,2,..., nrhs) and is set to zero otherwise.

2.2.4 To preserve a partial factorization

If the user wishes to terminate the sequence of calls to MA42_ELEMENT_FACTORIZE before all nelt elements have been entered, a call of the following form may be made. This will restore internal arrays to the values they had before the first call to MA42_ELEMENT_FACTORIZE, deallocate arrays that are no longer needed, and allow the user to access the data remaining in the front. Direct-access files are not closed or deleted.

```
CALL MA42_ELEMENT_PARTIAL(keep,info,control)
```

keep is a scalar of derived type MA42_ELEMENT_KEEP and INTENT(INOUT) that must be passed unchanged since the

last call to MA42_ELEMENT_FACTORIZE. The components of keep that may be of interest to the user after a call to MA42_ELEMENT_PARTIAL are described in Section 2.5.

info, control are as in the calls to MA42_ELEMENT_ANALYSE.

2.2.5 To solve further systems $\mathbf{AX}=\mathbf{B}$ or systems$\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ or $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$

After the final call to MA42_ELEMENT_FACTORIZE or after a call to MA42_ELEMENT_AFS (see Section 2.2.7), one or more calls of the following form may be made to solve $\mathbf{AX}=\mathbf{B}$, or $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$, or $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$. Only forward elimination or only back substitution may be performed by appropriately setting the optional parameter fbsolve.

```
CALL MA42_ELEMENT_SOLVE(nrhs,b,x,keep,info,control[,trans,fbsolve])
```

nrhs is an INTEGER scalar of INTENT(IN) that must be set to the number of right-hand sides. **Restriction:** nrhs $\geq$ 1.

b is a rank-2 array of package type, of INTENT(INOUT) and size at least info%lindex by nrhs. On entry, b must be set so that if I has been used to index a variable, b(I,J) is the corresponding component of the right-hand side for the J-th system (J = 1,2,..., nrhs). This array is changed if fbsolve is absent or is equal to 1. b is not accessed if fbsolve = 3.

x is a rank-2 array of package type of INTENT(INOUT) and size at least info%lindex by nrhs that is used to hold the solution. x needs to be set on entry only if the call follows a call to MA42_ELEMENT_PARTIAL. In this case, if fbsolve = 2 or 3 and K is the column index of a variable that is still in the front when the factorization was terminated (see lhed in Section 2.5), x(K,J) must hold the solution vector for variable K to the J-th system. In all cases, on exit, if I has been used to index a variable, x(I,J) holds the solution for variable I to the J-th system (J = 1,2,..., nrhs). Note that the uer can check whether a variable has been used to index a variable using keep%last (see Section 2.5).

keep, info, control are as in the calls to MA42_ELEMENT_ANALYSE.

trans is an optional INTEGER scalar of INTENT(IN). If not present, or if present and set to 1, $\mathbf{AX}=\mathbf{B}$ is solved, if trans = 2, $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ is solved, and if trans = 3, $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$ is solved. If fbsolve = 3, trans is ignored (transpose and conjugate systems cannot be solved in this case).

fbsolve is an optional INTEGER scalar INTENT(IN) that allows the user to specify that only forward eliminations or back substitutions are to be performed. If absent, $\mathbf{AX}=\mathbf{B}$ (or $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ or $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$) is solved. If fbsolve = 1, forward eliminations are performed, that is, $\mathbf{PLX}=\mathbf{B}$ (or $(\mathbf{UQ})^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ if trans = 2 or $(\mathbf{UQ})^{\mathrm{H}}\mathbf{X}=\mathbf{B}$ if trans = 3) is solved; if fbsolve = 2, back substitutions are performed, that is, $\mathbf{UQX}=\mathbf{B}$ (or $(\mathbf{PL})^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ if trans = 2 or $(\mathbf{PL})^{\mathrm{H}}\mathbf{X}=\mathbf{B}$ if trans = 3) is solved. If a partial factorization was performed with nrhs > 0 and preserved by a call to MA42_ELEMENT_PARTIAL, a partial back substitution may be performed (with nrhs unchanged since the calls to MA42_ELEMENT_FACTORIZE) by setting fbsolve = 3. **Restriction:** fbsolve = 1, 2, 3.

2.2.6 To compute the residual

A call of the following form must be made for each element. The elements may be passed in **any** order.

```
CALL MA42_ELEMENT_RESIDUAL(assem,nrhs,element,resid,x,keep,info, &
                           control[,trans,anrm])
```

assem is a LOGICAL scalar of INTENT(IN) that must be set to .true. if the call follows a call to MA42_ELEMENT_SOLVE and to .false. if it follows a call to MA42_ELEMENT_FACTORIZE.

nrhs is an INTEGER scalar of INTENT(IN) that must be set to the number of right-hand sides. **Restriction:** nrhs $\geq$ 1.

element is a scalar of derived type MA42_ELEMENT and INTENT(IN). On the call for the element matrix $\mathbf{A}^{[k]}$ the following components must be set:

element%nvar is an INTEGER scalar that must hold the number of variables in element k.

element%vars is a rank-1 INTEGER pointer array that must be allocated with size at least element%nvar and

set to hold the indices of the variables in element k.

element%mvar is an INTEGER scalar that must be at least as large as the largest number of variables in an element.

element%reals is a rank-2 pointer array of package type of size element%mvar by at least element%nvar that must contain the numerical values of the entries of the element matrix $\mathbf{A}^{[k]}$ in packed form.

If assem = .false., the following component must also be allocated and set:

element%rhs is a rank-2 pointer array of package type of size element%mvar by at least nrhs that must contain the element right-hand side $\mathbf{B}^{[k]}$ in packed form.

resid is a rank-2 array of package type, of INTENT(INOUT) and size at least info%lindex by nrhs. If assem = .true., resid must be set on entry so that if I has been used to index a variable, resid(I,J) is the corresponding component of the right-hand side for the J-th system (J = 1,2,..., nrhs). If assem = .false., resid need not be set on entry. In each case, on exit, resid(I,J) is the residual for variable I for the J-th system.

x is a rank-2 array of package type of INTENT(IN) and size at least info%lindex by nrhs. On entry, if I has been used to index a variable, x(I,J) must hold the solution for variable I to the J-th system (J = 1,2,..., nrhs).

keep, info, control are as in the calls to MA42_ELEMENT_ANALYSE.

trans is an optional INTEGER scalar of INTENT(IN) that indicates which system has been solved; x must hold the solution of this system. If trans is not present, or if present and set to 1, x must hold the solution to $\mathbf{AX}=\mathbf{B}$. If trans = 2 (respectively, 3), x must hold the solution to $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$ (respectively, $\mathbf{A}^{\mathrm{H}}\mathbf{X}=\mathbf{B}$). If assem = .false., trans is ignored and x must hold the solution to $\mathbf{AX}=\mathbf{B}$. trans must be unchanged between calls.

anrm is an optional scalar of real type of INTENT(INOUT). It need not be set on the first call but must be unchanged by the user between calls. On exit from the final call, if x holds the solution of $\mathbf{AX}=\mathbf{B}$, anrm holds an upper bound on the infinity norm of $\mathbf{A}$. If trans = 2 or 3, anrm holds an upper bound on the infinity norm of $\mathbf{A}^{\mathrm{T}}$ (which is equal to an upper bound on the infinity norm of $\mathbf{A}^{\mathrm{H}}$).

2.2.7 To analyse, factorize and solve using a single call

To perform the analyse, factorize, and (optionally) solve a system of finite-element equations and compute the corresponding residuals, a single call of the following form may be made after a call to MA42_ELEMENT_START.

```
CALL MA42_ELEMENT_AFS(n,nelt,eltvar,eltptr,lvalues,values,nrhs, &
          rhsval,order,x,keep,info,control[,resid,anrm,mfrelt,  &
          valnam,rhsnam])
```

n is an INTEGER scalar of INTENT(IN) that must be set to be at least as large as the largest integer used to index a variable. **Restriction:** n ≥ 1.

nelt is an INTEGER scalar of INTENT(IN) that must be set to the total number of finite elements in the problem. **Restriction:** nelt ≥ 1.

eltvar is a rank-1 INTEGER array of INTENT(IN) that must be set by the user to the variable indices belonging to the elements. The variables belonging to element 1 must precede those for element 2, and so on. If duplicate indices are detected in an element or variable indices less than 1 or greater than n are found, an error is returned (see -1 and -6 in Section 2.6). **Restriction:** 1 ≤ eltvar(:) ≤ n.

eltptr is a rank-1 INTEGER array of INTENT(IN) and size at least nelt+1. eltptr(IELT) must contain the position in eltvar of the first variable in the IELT-th element in the element variable lists (IELT = 1, 2,..., nelt), and eltptr(nelt+1) must be set to the position after the last variable in the last element.

lvalues is an INTEGER scalar of INTENT(IN). If the user wants the element matrix entries and element right-hand

sides to be read from files, lvalues must be set to 1. Otherwise, the entries are input using values and rhsval.

values is a rank-1 array of package type and INTENT(IN). If lvalues≠1, values must contain the numerical values of the element matrices $\mathbf{A}^{[k]}$. The $\mathbf{A}^{[k]}$ must be in the order given by eltvar and each must be held as a full matrix, stored by columns and the rows of each element matrix must be in the same order as in eltvar. This array is not accessed if lvalues=1.

nrhs is an INTEGER scalar of INTENT(IN) that must be set to the number of right-hand sides. If nrhs≤0, only the analyse and factorize phases are performed.

rhsval is a rank-1 array of package type and INTENT(IN). If lvalues≠1, rhsval must contain the element right-hand sides $\mathbf{B}^{[k]}$. The $\mathbf{B}^{[k]}$ must be in the order given by in eltvar and each must be held as a full matrix, stored by columns, and the rows of each $\mathbf{B}^{[k]}$ matrix must be in the same order as in eltvar. This array is not accessed if lvalues=1 or nrhs≤0.

order is a rank-1 INTEGER array of INTENT(INOUT) and size nelt. order must be set on entry if control%order63=0 (the default value is 6). In this case, order must hold the element assembly order. For other values of control%order63, a suitable element assembly order is computed by the package (see control%order63 in Section 2.3 for more details). On exit, order holds the order in which the elements were assembled.

x is a rank-2 array of package type and of INTENT(OUT). If nrhs>0, x must be of size at least n by nrhs. On exit, if I has been used to index a variable, x(I,J) holds the solution for variable I to system J and is set to zero otherwise (I = 1,2,..., n, J = 1,2,..., nrhs). x is not accessed if nrhs≤0.

keep is a scalar of derived type MA42_ELEMENT_KEEP and INTENT(INOUT). If the user wishes to solve for further right-hand sides or to solve transpose or conjugate systems, keep must be passed unchanged to MA42_ELEMENT_SOLVE.

info is a scalar of derived type MA42_ELEMENT_INFO and INTENT(OUT). Its components provide information about the execution of the subroutine, as explained in Section 2.4.

control is a scalar of derived type MA42_ELEMENT_CONTROL and INTENT(IN). Its components control the action, as explained in Section 2.3.

resid is an optional rank-2 array of package type and of INTENT(OUT). If present and nrhs>0, resid must be of size at least n by nrhs. On exit, if I has been used to index a variable, resid(I,J) holds the residual for variable I for system J and is set to zero otherwise (I = 1,2,..., n, J = 1,2,..., nrhs). resid is not accessed if nrhs≤0.

anrm is an optional scalar of real type of INTENT(OUT). On exit, if resid and anrm are both present, anrm holds an upper bound on the infinity norm of **A**. anrm is not accessed if either nrhs≤0 or resid is not present.

mfrelt is an optional scalar of type INTEGER and INTENT(IN) that must be present if lvalues=1. If present, mfrelt determines the record length given in the RECL= specifier of the OPEN statements for the direct-access files for the element data. mfrelt must be at least as large as the largest number of variables in an element (if not, the error -26 is returned, see Section 2.6).

valnam is an optional scalar of type CHARACTER, length at most 128, and INTENT(IN) that must be present if lvalues=1. If present, valnam must contain the name of the direct-access file holding the element matrices $\mathbf{A}^{[k]}$. The record length given in the RECL= specifier of the OPEN statement for the direct-access file must be that required for arrays of package type and size mfrelt**2. The *i*-th record must hold the element matrix corresponding the *i*-th element in eltvar. Each $\mathbf{A}^{[k]}$ must be held as a full matrix stored by columns and the rows of each element matrix must be in the same order as in eltvar.

rhsnam is an optional scalar of type CHARACTER, length at most 128, and INTENT(IN) that must be present if lvalues=1 and nrhs>0. If present, rhsnam must contain the name of the direct-access file holding the

element right-hand matrices $\mathbf{B}^{[k]}$. The record length given in the RECL= specifier of the OPEN statement for the direct-access file must be that required for arrays of package type and size mfrelt*nrhs. The *i*-th record must hold the element right-hand side matrix corresponding to the *i*-th element in eltvar ($i = 1,2,...,$nelt). Each $\mathbf{B}^{[k]}$ must be held as a full matrix, stored by columns, and the rows of each $\mathbf{B}^{[k]}$ matrix must be in the same order as in eltvar.

2.2.7 The terminating subroutine

A single call of the following form should be made after the last call for a particular problem to other routines in the module:

```
CALL MA42_ELEMENT_FINAL(element,keep,control)
```

element is a scalar of derived type MA42_ELEMENT and INTENT(INOUT). On exit, its components that are pointer arrays will have been deallocated.

keep is a scalar of derived type MA42_ELEMENT_KEEP and INTENT(INOUT). On exit, its components that are pointer arrays will have been deallocated.

control is a scalar of derived type MA42_ELEMENT_CONTROL and INTENT(IN). Its components control the action, as explained in Section 2.3.

2.3 Derived data type for control of the subroutines

The module contains a derived type called MA42_ELEMENT_CONTROL which has the following components. Default values are set by the cal to MA42_ELEMENT_START.

Printing controls

lp is an INTEGER scalar that holds the unit number for error messages. Printing is suppressed if lp < 0. The default value is 6.

wp is an INTEGER scalar that holds the unit number for warning messages. Printing is suppressed if wp < 0. The default value is 6.

mp is an INTEGER scalar that holds the unit number for diagnostic printing. Printing is suppressed if mp < 0. The default value is 6.

print_level is an INTEGER scalar indicating the level of diagnostic printing desired. Possible values are:

< 0: no printing.

0: error and warning messages only.

1: as 0 plus basic diagnostic printing.

> 1: as 1 plus some more detailed diagnostic messages.

The default value is 0.

Other controls (in alphabetical order)

alpha is a scalar of real type with default value 0.01. An entry of the frontal matrix is only considered suitable for use as a pivot if it is of absolute value at least as large as alpha times the entry of largest absolute value in its column.

front_multiple is a scalar of real type with default value 1.1. If at any point in the factorization the frontsize is found to be too small, it is increased to front_multiple times the minimum size that will allow the computation to continue. If front_multiple is less than or equal to 1.0, the computation will terminate as soon as the frontsize is found to be too small (see error -12 in Section 2.6). Further details are given in Section 2.7.

lbuffer is a LOGICAL scalar with default value .true. that controls the action if the user-supplied lenbuf values

are too small and filnam was not present on the first call to MA42_ELEMENT_FACTORIZE. If lbuffer is set to .true., scratch files will be used to allow the computation to continue. Otherwise, the computation terminates with an error (see error -14 in Section 2.6). lbuffer is not used if filnam is present.

ldelete is a LOGICAL scalar that controls whether or not the direct-access files used to hold the factors are deleted on a call to MA42_ELEMENT_FINAL. If ldelete = .true., the files are deleted (note that files with status SCRATCH are always deleted). The default value is .false..

ldiag is a LOGICAL scalar that controls whether or not the pivoting is restricted to the diagonal. If ldiag = .true., pivoting is restricted to the diagonal until the last element has been entered and no diagonal pivots can be chosen. At this point, off-diagonal pivots are used. The default value is .false..

lfactor is a LOGICAL scalar with default value .true.. In this case, the **L** factor is stored. If the user does not wish to store the **L** factor, lfactor should be set to .false. (**L** must be stored if the user wishes to call MA42_ELEMENT_SOLVE).

lsingular is a LOGICAL scalar with default value .true.. In this case, if **A** is found to be singular, a warning is issued and the factorization continues. Otherwise, the computation is terminated with an error (see error -27 in Section 2.6).

order63 is an INTEGER scalar with default value 6 that is used to select the algorithm for computing an element assembly order. Possible values are:

- 0: no reordering and the user must supply the element assembly order using the array order.
- 1: both the Sloan direct and indirect element orderings are computed and the one that produces the smallest root mean squared frontsize is selected.
- 2: the Sloan direct element ordering is used.
- 3: the Sloan indirect element ordering is used.
- 4: both the hybrid spectral-Sloan direct and indirect element orderings are computed and the one that produces the smallest root mean squared frontsize is selected.
- 5: the hybrid spectral-Sloan direct element ordering is used.
- 6: the hybrid spectral-Sloan indirect element ordering is used.

In the current version, all other values will lead to an error return -7. Note that the hybrid spectral-Sloan algorithms are more expensive (in terms of CPU time) than the Sloan algorithms but for large problems generally give smaller frontsizes.

pivot_size is an INTEGER scalar with default value 16. At each stage of the factorization, eliminations are only performed when there at least pivot_size pivot candidates (fully summed variables). Increasing pivot_size in general increases the number of floating-point operations and real storage requirements but allows greater advantage to be taken of Level 3 BLAS and reduces integer storage. The best value is both problem and machine dependent.

small is a scalar of real type with default value zero. The matrix is declared singular if, during the factorization, the entry of largest absolute value in any column of the reduced matrix is less than or equal to small.

2.4 Information returned to the user

The module contains a derived type called MA42_ELEMENT_INFO. The following components (in alphabtical order) are used to return information to the user.

deficiency is an INTEGER scalar that holds an estimate of the deficiency of the matrix **A**.

det is a scalar of real type that holds the natural logarithm of the modulus of the determinant of the matrix **A** (see also sign_det). If the matrix is found to be singular, det is set to zero.

file_bound is a rank-1 INTEGER(long) array of size 3. file_bound(1:3) hold estimates of the storage needed (in words) for the factors **U** and **L**, and the integer factor data computed by the analyse phase.

flag is an INTEGER scalar that is used as an error flag. If a call is successful, flag has value 0 on exit. A nonzero value of indicates an error has been detected or a warning issued (see Section 2.6). If an error is detected, the information contained in other components of info may be incomplete.

flagold is an INTEGER scalar. On each exit with flag = -3, flagold holds the value of flag that was returned on the previous call.

flag63 is an INTEGER scalar that holds the error flag returned by MC63 (control%order63 = 1, 2, or 3). If MC63 has not been called, it is set to zero.

flag73 is an INTEGER scalar that holds the error flag returned by HSL_MC73 (control%order63 = 4, 5, or 6). If HSL_MC73 has not been called, it is set to zero.

flops is a scalar of real type that holds the number of floating-point operations in the innermost loops of the eliminations. This count includes operations performed during static condensation.

front_bound is an INTEGER scalar that holds the estimate of the maximum frontsize computed by the analyse phase. Note that in general this estimate is a lower bound on the maximum frontsize required by the factorization.

iostat is an INTEGER scalar that holds the Fortran iostat parameter (the iostat parameter is a machine and complier-dependent parameter that is set to zero if a Fortran READ, WRITE, OPEN, or INQUIRE statement is correctly executed and to a positive integer otherwise). See error returns -15 and -17 in Section 2.6.

lindex is an INTEGER scalar that holds the largest index used by the user to index a variable.

max_buffers is a rank-1 INTEGER array of size 3 that holds the maximum number of buffers used to hold a block of pivot rows, a block of pivot columns, and the integer data for a block of pivot rows and columns.

max_front is an INTEGER scalar that holds the maximum frontsize (that is, the order of the largest frontal matrix). If error -12 has been returned, this value may only be a lower bound.

mvar is an INTEGER scalar that holds the largest number of variables in an element.

ndf is an INTEGER scalar that holds the total number of variables in the problem. This will be less than lindex if the variables are not numbered contiguously.

nonzeros is a rank-1 INTEGER(long) array of size 2 that holds the number of entries in the **U** and **L** factors.

pivot is a rank-1 INTEGER array of size 3 that is used to return information relating to the pivots chosen. pivot(1:2) hold the number of diagonal and off-diagonal pivots, respectively, and pivot(3) holds the size of the largest pivot block used.

sign_det is an INTEGER scalar. In the double precision version, sign_det is set to +1 (-1) if the determinant of the matrix is positive (negative). If the matrix is found to be singular, sign_det is set to 0. In the double complex versions, sign_det is set to +1 if the matrix is non-singular and to 0 if it is singular. (See also det).

stat is an INTEGER scalar that holds the Fortran stat parameter (the stat parameter is a machine and complier-dependent parameter that is set to zero if a Fortran ALLOCATE statement is correctly executed and to a positive integer otherwise). See error return -16 in Section 2.6.

static is a rank-1 INTEGER array of size 2 that is used to return information relating to static condensation. static(1:2) hold, respectively, the number of static condensations performed and the number of potential static condensations. The number of potential static condensations will exceed the number performed if numerical considerations prevent immediate elimination of variables internal to an element.

storage is a rank-1 INTEGER(long) array of size 3 that holds the number of entries stored for the **U** factor (this includes the corresponding right-hand sides), the **L** factor, and the integer factor data. Note that

storage(2)=nonzeros(2) and, if nrhs=0, storage(1)=nonzeros(1).

stream is a rank-1 INTEGER array of size 5. stream(1:3) hold the unit numbers used by the direct access files for the **U** factor, the **L** factor, and the integer factor data. If direct access files are not used, stream is set to zero. If MA42_ELEMENT_AFS is called with lvalues=1, stream(4:5) hold the unit numbers used to read in the element matrices and element right-hand side matrices, respectively.

strm is an INTEGER scalar that is set to a nonzero value if the error –15 is returned (see Section 2.6). In this case, strm holds the unit number for which the OPEN statement failed.

time_analyse is a REAL scalar that holds the processor time in seconds taken by the analyse phase (this includes, when applicable, the time to reorder the elements).

time_factor is a REAL scalar that holds the processor time in seconds taken to perform the matrix factorization.

time_factorb is a REAL scalar that holds the processor time in seconds taken to perform the matrix factorization and solve the linear systems when element right-hand sides are specified in the calls to MA42_ELEMENT_FACTORIZE.

time_order is a REAL scalar that holds the processor time in seconds taken to reorder the elements.

time_solve is a REAL scalar that holds the processor time in seconds taken to solve the linear system(s), that is, the time taken by MA42_ELEMENT_SOLVE.

time_residual is a REAL scalar that holds the processor time in seconds taken to compute the residuals (and, optionally, an upper bound on the norm of the system matrix).

2.5 Derived type for passing information between the subroutines and for preserving a partial factorization

The module contains a derived type called MA42_ELEMENT_KEEP that must be passed unchanged by the user between all calls to routines in the module for a particular problem. The following components may be of interest to the user (in particular, they may be useful after a call to MA42_ELEMENT_PARTIAL).

nfvar is a scalar of type INTEGER. On exit from each call to MA42_ELEMENT_FACTORIZE, nfvar holds the number of rows and columns remaining in the front.

khed is a rank-1 INTEGER pointer array. On exit from each call to MA42_ELEMENT_FACTORIZE, khed(1:nfvar) holds the row indices of the variables remaining in the front.

lhed is a rank-1 INTEGER pointer array. On exit from each call to MA42_ELEMENT_FACTORIZE, lhed(1:nfvar) holds the column indices of the variables remaining in the front.

fa is a rank-2 pointer array of package type. On exit from each call to MA42_ELEMENT_FACTORIZE, fa(1:nfvar,1:nfvar) holds the frontal matrix.

frhs is a rank-2 pointer array of package type. If nrhs>0, on exit from each call to MA42_ELEMENT_FACTORIZE, frhs(1:nfvar,1:nrhs) holds the frontal right-hand sides.

last is a rank-1 INTEGER pointer array. On exit from the final call to MA42_ELEMENT_ANALYSE (and on exit from MA42_ELEMENT_AFS), if I has been used to index a variable, last(I) is the assembly step at which variable I appears for the last time. If I is not used to index a variable, last(I) is set to zero (I = 1, 2, ..., info%lindex). On exit from the final call to MA42_ELEMENT_FACTORIZE, or on exit from a call to MA42_ELEMENT_PARTIAL, last is restored to its value on exit from the final call to MA42_ELEMENT_ANALYSE.

2.6 Error diagnostics

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 will be output on unit control%lp. Possible negative values are:

–1 One or more of the indices in an element variable list are out of range. (ANALYSE, FACTORIZE, AFS, and RESIDUAL).

–2 Either n < 1 or nelt < 1 (ANALYSE first call only and AFS).

–3 An error was returned on an earlier call. The error flag from the earlier call is held in info%flagold. (ANALYSE, FACTORIZE, PARTIAL, SOLVE, AFS, and RESIDUAL). This error is also returned if the user's first call to FACTORIZE does not follow calls to ANALYSE.

–4 The number of variables in at least one element is less than 1. (ANALYSE, FACTORIZE, AFS, and RESIDUAL).

–5 Either the array element%vars has not been allocated or it is too small (ANALYSE, FACTORIZE, and RESIDUAL).

–6 Duplicate occurrences of the same variable index found in an element. (ANALYSE and AFS).

–7 Either control%order63 has an invalid value or there is an unexpected error return from the element ordering routine. The error flag returned by MC63 (HSL_MC73) is held in info%flag63 (info%flag73). This error is also returned if control%order = 0 and order does not contain a permutation. (ANALYSE and AFS).

–8 element%mvar is too small, or element%reals or element%rhs has not been allocated, or there is an error in the size of element%reals or element%rhs. (FACTORIZE and RESIDUAL).

–9 Error detected when writing to a direct-access file. The iostat parameter is returned in info%iostat. (FACTORIZE, PARTIAL, and AFS).

–10 Error detected when reading from a direct-access file. The iostat parameter is returned in info%iostat. (FACTORIZE, SOLVE, and AFS).

–11 Failed to find a unit to which a file could be connected. (ANALYSE, FACTORIZE, and AFS).

–12 frontsize too small and control%front_multiple ≤ 1. (FACTORIZE).

–13 Either an index list for an element has been changed since ANALYSE was called or the assembly order of the elements differs from that given by order (FACTORIZE).

–14 lenbuf(I) < 0 (I = 1, 2, or 3) (FACTORIZE first call only). This error is also returned if control%lbuffer is set to .false. and the user supplied lenbuf is found to be too small. (FACTORIZE).

–15 Error in a Fortran OPEN statement. The iostat parameter is returned in info%iostat and the unit number for which the error occurred is returned in info%strm. (ANALYSE, FACTORIZE, SOLVE, and AFS).

–16 Error in a Fortran ALLOCATE statement. The stat parameter is returned in info%stat. If direct-access files have not been used, it may be possible to avoid this error by reducing lenbuf and rerunning using files. (ANALYSE, FACTORIZE, SOLVE, and AFS).

–17 Error in a Fortran INQUIRE statement. The iostat parameter is returned in info%iostat. (ANALYSE, FACTORIZE, and AFS).

–18 Either the array x is not present when expected (nrhs > 0) or x is of incorrect size. (FACTORIZE final call only, SOLVE, and AFS). If this error is returned on the final call to FACTORIZE, the factorization will have been performed correctly and the user may go on to call SOLVE.

–19 SOLVE or RESIDUAL has been called after a call to FINAL for the same problem.

–20 fbsolve has an invalid value. (SOLVE).

–21 nrhs ≤ 0. (SOLVE and RESIDUAL). This error is also returned by SOLVE if fbsolve = 3 and the value of nrhs is not the same as it was on the first call to FACTORIZE. It is also returned by RESIDUAL if assem = .false. and the value of nrhs is greater than it was on the first call to FACTORIZE.

–22 b is of incorrect size (either size(b,1) < info%lindex or size(b,2) < nrhs) (fbsolve ≠ 3). (SOLVE). This error is also returned if resid is of incorrect size (RESIDUAL and AFS).

–23 SOLVE called but the **L** factor has not been stored (control%lfactor = .false.).

–24 PARTIAL called either after a complete set of calls to FACTORIZE or before any calls have been made to

FACTORIZE.

–25 lvalues ≠ 1 and either values is of size less than $\sum nvar(k)^2$ or rhsval is of size less than $\sum(nvar(k)*nrhs)$ (where $nvar(k)$ is the number of variables in element k and the summation is over all elements) (AFS). This error is also returned by AFS if lvalues = 1 and either valnam or rhsnam is not present.

–26 lvalues = 1 but either mfrelt is not present or mfrelt < info%mvar (AFS).

–27 **A** found to be singular and control%lsingular = .false. (FACTORIZE and AFS).

A positive value of info%flag is associated with a warning message that will be output on unit control%wp. The warnings are:

+2 Ordering has not reduced the root mean squared wavefront and so the original input element order is retained (that is, order(i) = i, i = 1, 2, ..., nelt). (ANALYSE final call only and AFS).

+4 **A** found to be singular and control%lsingular = .true. (FACTORIZE and AFS).

+8 frontsize (either the user-supplied value or the default if frontsize is not present) is too small and control%front_multiple > 1. The computation continues but will NOT be as efficient as having frontsize large enough. The minimum value for frontsize that will avoid this warning on a subsequent run with identical data is returned in info%max_front (see Section 2.4). (FACTORIZE and AFS).

+16 This warning is returned on the first call to FACTORIZE if the user-supplied values of lenbuf are too large to allocate in-core buffers of the requested length. Default buffer sizes are used (see Section 2.8) and, if filnam is not present, direct-access files with status SCRATCH are used to hold the factors. This warning can also be returned on a later call to FACTORIZE if control%lbuffer = .true. and lenbuf found to be too small. In this case, the factors will be held using one or more direct access files with status SCRATCH. (FACTORIZE).

+32 This warning is returned on the first call to RESIDUAL or on a call to AFS if an allocation error occurs when computing anrm (an array of real type and length info%lindex is required). In this case, resid is still computed but anrm is set to zero. (RESIDUAL and AFS).

Positive values of flag are summed so that the user can identify all warnings issued, e.g. flag = 10 indicates both warnings 2 and 8 are raised.

2.7 Choosing the maximum frontsize

On exit from the final call to MA42_ELEMENT_ANALYSE, info%front_bound holds an estimate of the maximum frontsize. If frontsize is not present on the first call to MA42_ELEMENT_FACTORIZE, the code will choose a maximum frontsize for the factorization of 1.1*info%front_bound. The extra space is to allow for possible increases to the frontsize because of delayed pivots. If during the computation this frontsize (or the user-supplied frontsize) is too small, the action taken depends upon control%front_multiple (see Section 2.3). To allow the computation to continue, control%front_multiple must be greater than 1. In this case, if the frontsize is found to be too small, the contents of the internal arrays of size that depends on the maximum frontsize are written to scratch files, the arrays are deallocated and then reallocated with sizes sufficient to continue the computation. The data in the files is read back into these arrays, the scratch files are closed (and thus deleted), and the computation continues. Note that increasing the size of internal arrays will add to the factorization cost and may be done more than once during the factorization of a particular problem. If control%front_multiple is less than or equal to 1, the computation will terminate with the error flag -12 as soon as the frontsize is found to be too small. The user may increase frontsize and restart the series of calls to MA42_ELEMENT_FACTORIZE (there is no need to repeat the calls to MA42_ELEMENT_ANALYSE).

2.8 Choosing buffer sizes and using direct-access files

In HSL_MA42_ELEMENT, three direct-access files may be used: one for the **U** factor (which is held with the corresponding right-hand sides), one for the **L** factor, and one for the row and column indices of the variables in the factors. The user can choose whether the factor data is held in main memory (in so-called buffers) or written to

direct-access files. The user's choice will normally depend upon the problem size and memory available. If the user wishes to hold the data in main memory, the optional argument lenbuf should be present on the first call to MA42_ELEMENT_FACTORIZE. In this case, the user should choose buffer lengths based on the information returned by MA42_ELEMENT_ANALYSE in info%file_bound. We advise lenbuf(1) should be at least info%file_bound(1) + info%lindex*nrhs, lenbuf(2) be at least info%file_bound(1), and lenbuf(3) be at least 1.1*info%file_bound(2). If the user-supplied values are too small, the action taken depends on control%lbuffer. If set to .true. (the default), scratch files are used to allow the computation to continue. Otherwise, the computation terminates with the error -14.

If lenbuf is not present, the default values used by the code are 2^{16}; the default values are also used by MA42_ELEMENT_AFS. In this case, direct-access files may be needed to hold the factors. If filnam is not present on the first call to MA42_ELEMENT_FACTORIZE, these files will have status SCRATCH. The default values for lenbuf are also used if the user-supplied values are so large that they lead to an allocation error (see warning +16 in Section 2.6).

If files are used, each record in the **U** and **L** files will hold lenbuf(1) and lenbuf(2) scalars of package type, respectively, and each record in the integer data file will hold lenbuf(3) scalars of type INTEGER. If control%lfactor = .false., the **L** factor is not stored and lenbuf(2) is not accessed.

3 GENERAL INFORMATION

3.1 Summary of information.

Other routines called directly: The HSL routines MC63AD and HSL_MC73_DOUBLE. The BLAS routines IDAMAX, DAXPY/ZAXPY, DGER, ZGERU, DGEMV/ZGEMV, DTPSV/ZTPSV, DGEMM/ZGEMM, DTRSM/ZTRSM.

Input/output: In the event of errors, diagnostic messages are printed. The output units for these messages are controlled by control%lp, control%mp, and control%wp (see Section 2.3).

Restrictions:

n ≥ 1 (ANALYSE first call only and AFS).
nelt ≥ 1 (ANALYSE and AFS).
element%nvar ≥ 1 (ANALYSE, FACTORIZE, and RESIDUAL).
element%vars(:) ≥ 1 (ANALYSE, FACTORIZE, and RESIDUAL).
element%vars(:) ≤ n (ANALYSE).
element%vars(:) ≤ info%lindex (FACTORIZE and RESIDUAL).
element%mvar(:) ≤ info%mvar (FACTORIZE and RESIDUAL).
nrhs ≥ 1 (SOLVE and RESIDUAL).
1 ≤ eltvar(:) ≤ n (AFS).
0 ≤ control%order63 ≤ 6 (ANALYSE and AFS).

Optional arguments:

lenbuf(:) ≥ 1 (FACTORIZE first call only).
fbsolve = 1, 2 or 3 (SOLVE).

Portability: Fortran 95.

Major changes: Version 2.0.0: Kind of lenbuf argument to MA42_ELEMENT_FACTORIZE changed from long to default integer.

4 METHOD

The method is based on that used by the earlier HSL package MA42 (see Duff and Scott 1996 and the references therein). The elements are assembled into a frontal matrix one at a time. A variable that has appeared for the last time (i.e. does not occur in future elements) is fully summed and is available for use as a pivot in the Gaussian elimination. Elements are assembled into the frontal matrix until there are at least control%pivot_size fully summed variables (pivot candidates).

Eliminations are performed using a pivot candidate provided it satisfies a numerical tolerance. Once all possible eliminations for the current stage have been performed, the pivot rows and, optionally, the pivot columns are written to buffers and thence, if requested (or if the buffers are too small), to direct-access files. While elements remain unassembled, further assembles into the frontal matrix are then performed until there are again at least control%pivot_size pivot candidates.

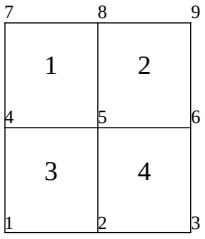
For efficiency (in terms of flop counts and memory required), the elements need to be ordered so that the same variable does not occur in elements that are widely apart in the ordering. The user can supply a suitable ordering; otherwise, MA42_ELEMENT_ANALYSE and MA42_ELEMENT_AFS use the HSL routines MC63 and HSL_MC73 to obtain an efficient ordering.

References.

Duff, I.S. and Scott, J.A. (1996) The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Softw.*, **22**, 30-45.

5 EXAMPLE OF USE

Consider the following simple finite-element problem in which the finite-element mesh comprises four 4-noded quadrilateral elements with one degree of freedom at each node i, $1 \le i \le 6$ (the nodes 7, 8, and 9 are assumed constrained).



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

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

where the variable indices are indicated by the integers before each matrix (columns are identified symmetrically to rows). The corresponding element right-hand sides $\mathbf{B}^{[k]}$ $(1 \le k \le 4)$ are

$$\begin{pmatrix} 3. \\ 6. \end{pmatrix} \qquad \begin{pmatrix} 5. \\ 12. \end{pmatrix} \qquad \begin{pmatrix} 12. \\ 5. \\ 12. \\ 11. \end{pmatrix} \qquad \begin{pmatrix} 14. \\ 8. \\ 17. \\ 14. \end{pmatrix}.$$

The following program may be used to solve this problem. In this program we read the element data into arrays eltptr (location of first variable in element), eltvar (variable indices), value (numerical values), and rhs (right-hand sides). This method of storing the element data is used here for illustrative purposes; the user may prefer, for example, to read in the element data from a direct-access file or to generate the element data as it is required.

```
      PROGRAM EXAMPLE
      USE HSL_MA42_element_double

      integer, parameter  :: nelt = 4
      TYPE (MA42_ELEMENT) :: element
      TYPE (MA42_CONTROL) :: control
      TYPE (MA42_INFO)    :: info
      TYPE (MA42_KEEP)    :: keep
      integer :: ielt,jelt,jstrt,kstrt,l,n,nvar,nz,maxe,rhscrd,valcrd
      integer :: eltptr(nelt+1),valptr(nelt+1),order(nelt)
      integer,allocatable  :: eltvar(:)
      double precision, allocatable :: rhsval(:),values(:),x(:,:)

      call MA42_ELEMENT_START(element,keep,control)

! Read in the element variable lists.
! nz: total number of entries in the element lists
! maxe: largest number of variables in one element.
      read (5,*) n,nz,maxe
      allocate (eltvar(nz))
      read (5,*) eltptr; read (5,*) eltvar

      allocate (element%vars(maxe))
      allocate (element%reals(maxe,maxe))
      allocate (element%rhs(maxe,1))
      element%mvar = maxe
! Analyse and set valptr to point to start of reals
      valptr(1) = 1
      do ielt = 1,nelt
         nvar = eltptr(ielt+1) - eltptr(ielt)
         jstrt = eltptr(ielt)
         element%nvar = nvar
         element%vars(1:nvar) = eltvar(jstrt:jstrt+nvar-1)
         call MA42_ELEMENT_ANALYSE(n,nelt,element,order,keep,info,control)
         if (info%flag < 0) go to 60
         valptr(ielt+1) = valptr(ielt) + nvar*nvar
      end do

! Allocate solution array x
      allocate (x(info%lindex,1))

! Input element matrices and right-hand sides.
      read (5,*) valcrd, rhscrd
      allocate (values(valcrd));  allocate (rhsval(rhscrd))
      read (5,*) values;  read (5,*) rhsval

! Factorize and solve for single right-hand side
      do jelt = 1,nelt
         ielt = order(jelt)
         nvar = eltptr(ielt+1) - eltptr(ielt)
         element%nvar = nvar
         jstrt = eltptr(ielt)
         element%vars(1:nvar) = eltvar(jstrt:jstrt+nvar-1)
         kstrt = valptr(ielt)
         do l = 1,nvar
            element%reals(1:nvar,l) = values(kstrt:kstrt+nvar-1)
            kstrt = kstrt + nvar
         end do
         element%rhs(1:nvar,1) = rhsval(jstrt:jstrt+nvar-1)
         call MA42_ELEMENT_FACTORIZE(1,element,keep,info,control,x=x)
         if (info%flag < 0) go to 60
```

```
        end do

      write (6,'(/a)') 'The solution is:'
      write (6,'(6g12.4)') x(1:info%lindex,1)
! Deallocate arrays.
      deallocate (eltvar,rhsval,values,x)
      go to 70
   60 write (6,'(/a)') 'Unexpected error return'
   70 call MA42_ELEMENT_FINAL(element,keep,control)

      END PROGRAM EXAMPLE
```

The input data used for this problem is:

```
     6   12    4
     1    3    5    9   13
     4    5    5    6    4    5    1    2    5    6    2    3
    40   12
     2.  -1.   1.   7.   3.   4.   2.   8.   4.  -1.   2.   3.   3.
     1.   3.   2.   2.   3.   6.   1.   3.   2.   1.   5.   2.   1.
     8.   3.   1.   3.   2.   2.   8.   2.   2.   5.   3.   2.   5.
     4.
     3.   6.   5.  12.  12.   5.  12.  11.  14.   8.  17.  14.
```

This produces the following output:

```
 The solution is:
   1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
```

We could also solve this problem using MA42_ELEMENT_AFS as follows:

```
      PROGRAM EXAMPLE2
      USE HSL_MA42_element_double

      integer, parameter  ::  nelt = 4
      TYPE (MA42_ELEMENT) ::  element
      TYPE (MA42_CONTROL) ::  control
      TYPE (MA42_INFO)    ::  info
      TYPE (MA42_KEEP)    ::  keep
      integer :: n,nrhs,nz,rhscrd,valcrd
      integer :: eltptr(nelt+1),order(nelt)
      integer,allocatable  :: eltvar(:)
      double precision ,allocatable :: rhsval(:),values(:),x(:,:)

     call MA42_ELEMENT_START(element,keep,control)

! Read in the element integer data.
      read (5,*) n,nz
      allocate (eltvar(nz))
      read (5,*) eltptr;  read (5,*) eltvar

! Input element matrices and right-hand sides.
      nrhs = 1
      read (5,*) valcrd,rhscrd
      allocate (values(valcrd)); allocate (rhsval(rhscrd))
      read (5,*) values;  read (5,*) rhsval

      allocate (x(n,nrhs))
      call MA42_ELEMENT_AFS(n,nelt,eltvar,eltptr,0,values,nrhs, &
                            rhsval,order,x,keep,info,control)
      if (info%flag < 0) go to 60

      write (6,'(/a)') 'The solution is:'
```

```
      write (6,'(6g12.4)') x(1:info%lindex,1)
  ! Deallocate arrays.
      deallocate (eltvar,rhsval,values,x)
      go to 70
   60 write (6,'(/a)') 'Unexpected error return'
   70 call MA42_ELEMENT_FINAL(element,keep,control)

      END PROGRAM EXAMPLE2
```

With the same input file, we get the same output as before.