1 SUMMARY

The module HSL_MP62 **uses the multiple front method to solve sets of symmetric positive definite finite-element equations $\mathbf{AX}=\mathbf{B}$ that have been divided into non-overlapping subdomains**. The HSL routines MA62 and MA72 are used with MPI for message passing.

The coefficient matrix **A** must be of the form

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

where the summation is over finite elements. The element matrix $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns which correspond to variables in the k-th element. The right-hand side(s) **B** may optionally be in the form

$$\mathbf{B}=\sum_{k=1}^{m}\mathbf{B}^{(k)} \tag{2}$$

where $\mathbf{B}^{(k)}$ is nonzero only in those rows which correspond to variables in element k.

In the multiple front method, a frontal decomposition is performed on each subdomain separately. Thus, on each subdomain, a partial L factor is computed. Once all possible eliminations have performed within a subdomain, there remain the interface variables that are shared by more than one subdomain. If $\mathbf{F}_i$ is the remaining frontal matrix for subdomain i, and $\mathbf{C}_i$ is the corresponding right-hand side matrix, then the remaining problem is

$$\mathbf{FY}=\mathbf{C}, \tag{3}$$

where $\mathbf{F}=\sum_i\mathbf{F}_i$ and $\mathbf{C}=\sum_i\mathbf{C}_i$. By treating each $\mathbf{F}_i$ as an element matrix, the interface problem (3) is also solved by the frontal method. Once (3) has been solved, back-substitution on the subdomains completes the solution.

The element data and/or the matrix factors are optionally held in direct-access files.

ATTRIBUTES — **Version:** 2.1.0. (19 March 2020) **Types:** Real (single, double). **Remark:** Symmetric definite version of HSL_MP42. **Calls:** HSL_MP01, KB08, MA62, MA72, MC53, MC63. **Language:** Fortran 95 + TR 15581 (allocatable components). **Original date:** 19th March 2020. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

The module HSL_MP62 has five separate phases:

(1) Initialize

(2) Analyse

(3) Factorize

(4) Solve

(5) Finalize

Prior to calling the first phase, the user must choose the number of processes to run on and must initialize MPI by calling MPI_INIT on each process. The processes have rank 0, 1, 2,... The **host** is the process with rank zero. The user must also define an MPI communicator for the package. The communicator defines the set of processes to be used by MP62. Each phase must then be called in order by each process (although the solve phase is optional). Before calling each phase, the user must have completed all tasks with the defined communicator (and with any other communicator

that overlaps the MP62 communicator). During the factorize, the matrix factors are generated. If element right-hand side matrices are supplied (that is, right-hand sides of the form (2)), the solution is returned to the user at the end of the factorize phase. If the right-hand sides are only available in assembled form, or if the user wishes to use the matrix factors generated by the factorize phase to solve for further right-hand sides, the solve phase should be called. The finalize phase deallocates all arrays that have been allocated by the package and, optionally, deletes all direct-access files used to hold the matrix factors. The user may factorize more than one matrix at the same time by running more than one instance of the package; a instance of the package is terminated by calling the finalize phase. After the finalize phase and once the user has completed any other calls to MPI routines he or she wishes to make, the user should call MPI_FINALIZE to terminate the use of MPI. This is illustrated in the simple example code given in Section 5.

Access to the module requires a USE statement and the user must declare a structure data of derived type MP62_DATA defined by the module. HSL_MP62 has a single user-callable subroutine MP62A/AD with a single parameter data of type MP62_DATA. If the user wishes to run more than one instance of the module at once, a separate parameter of type MP62_DATA is needed for each instance.

The derived datatype has many components. In the following sections we describe the components of interest to the user. In Section 2.2, we describe the components that must be set by the user and that contain the solution vector, in Section 2.3 we describe the components that control the action, and in Section 2.4 we describe the components that hold information of potential interest to the user.

The following pseudocode illustrates how MP62A/AD must be used.

Single precision version

```
USE HSL_MP62_SINGLE
...
INTEGER ERCODE
TYPE (MP62_DATA) data
...
CALL MPI_INIT(ERCODE)
...
CALL MP62A (data)
...
CALL MPI_FINALIZE(ERCODE)
```

Double precision version

```
USE HSL_MP62_DOUBLE
...
INTEGER ERCODE
TYPE (MP62_DATA) data
...
CALL MPI_INIT(ERCODE)
...
CALL MP62AD (data)
...
CALL MPI_FINALIZE(ERCODE)
```

2.2 Input and output components

Prior to the first call to MP62A/AD for the current instance, the user must initialize the following component of data:

data%COMM is a scalar of type default INTEGER that must hold an MPI communicator. data%COMM must be initialized prior to the first call to MP62A/AD to define the set of processes to be used by MP62. Before each phase is called,

the user must have completed all tasks involving data%COMM (or involving any other communicator that overlaps data%COMM). It is not altered. Note that the code may be run using a single process.

For each call to MP62A/AD, a job parameter is needed. This parameter determines which phase of the package is to be performed.

data%JOB is a scalar of type default INTEGER that must be initialized on **all** processes before **each** call to MP62A/AD. It must be given the same value on all processes. It is not altered.

JOB = 1 initializes an instance of the module. A call with JOB = 1 must be made before any other calls to the module. On exit, the components of data that control the action contain default values. If the user wishes to use values other than the defaults, the corresponding components of data should be reset after the call with JOB = 1. Full details of the control components of data are given in Section 2.3.

JOB = 2 uses element variable lists to generate lists of interface variables and, optionally, to reorder the elements within each subdomain and allocate subdomains to processes.

JOB = 3 uses the information from the call with JOB = 2 to generate a partial L factor for each subdomain, form the factorization of the interface problem, and, if $\mathbf{B}^{(k)}$ are specified, to solve the equations $\mathbf{AX}=\mathbf{B}$ with right-hand side(s) $\mathbf{B}=\sum_{k=1}^{m}\mathbf{B}^{(k)}$. A call with JOB = 3 must be preceded by a call with JOB = 2.

JOB = 23 performs JOB = 2 then JOB = 3.

JOB = 4 uses the factors produced by a call with JOB = 3 to rapidly solve further systems of the form $\mathbf{AX}=\mathbf{B}$ ($\mathbf{B}$ in assembled form). A call with JOB = 4 must be preceded by a call with JOB = 3 (or 23) but several calls with JOB = 4 may follow a single call with JOB = 3 (or 23).

JOB = 5 deallocates all arrays that have been allocated by the module and, optionally, deletes all direct-access files that have been used to hold the matrix factors. A call with JOB = 5 should be made after all other calls for the current instance are complete. Note that components of data that are allocated by the user are **not** deallocated.

2.2.1 Input components for data%JOB = 2 or 23

Prior to a call with data%JOB = 2 or 23, the following components **must** be set by the user on the host (note that if data%JOB = 23, the input components described in Section 2.2.3 must also be set):

data%NDOM is a scalar of type default INTEGER that must be set to the number of subdomains. This component is not altered. **Restriction:** data%NDOM > 1.

data%NELT is a scalar of type default INTEGER that must be set to the number of elements in the problem. This component is not altered. **Restriction:** data%NELT ≥ data%NDOM.

data%NELTSB is a rank-1 allocatable array of type default INTEGER that must be allocated by the user with size at least data%NDOM. On entry, data%NELTSB(JDOM) must hold the number of elements in subdomain JDOM (JDOM = 1, 2,..., data%NDOM). This component is not altered. **Restriction:** data%NELTSB(:) > 0.

data%ELTVAR is a rank-1 allocatable array of type default INTEGER that must be allocated by the user and set to contain lists of the variable indices belonging to the finite elements. The lists of the variables in each of the elements belonging to subdomain 1 must precede those for the elements belonging to subdomain 2, and so on. If duplicate indices are detected in an element or variable indices less than 1 are found, the computation terminates with an error. This is component is not altered.

data%ELTPTR is a rank-1 allocatable array of type default INTEGER that must be allocated with size at least data%NELT+1. data%ELTPTR(IELT) must contain the position in data%ELTVAR of the first variable in the IELT-th element in the element variable lists (IELT = 1, 2,..., data%NELT), and data%ELTPTR(data%NELT+1) must be set to the position after the last variable in the last element. This component is not altered.

Additionally, the following component must be allocated and set if the control component data%ICNTL(10) (see Section 2.3) is nonzero.

data%INV_LIST is a rank-1 allocatable array of type default INTEGER that must be allocated with size at least data%NDOM. On entry, data%INV_LIST(JDOM) must hold the rank of the process that is to factorize subdomain JDOM (JDOM = 1, 2,..., data%NDOM). **Restriction:** $0 \leq$ data%INV_LIST(:) $<$ data%NPROC (where data%NPROC is the number of processes, see Section 2.4.1).

The following component must be allocated and set if data%ICNTL(9) > 0.

data%NORDER is a rank-2 allocatable array of type default INTEGER that must be allocated with size $\max_{1 \leq JDOM \leq data\%NDOM}$ data%NELTSB(JDOM)+1 by data%NDOM. It is used to hold the element assembly order. Within subdomain JDOM, the elements are locally labelled 1, 2,..., data%NELTSB(JDOM), according to the order in which the user inputs the variable lists in data%ELTVAR. On subdomain JDOM, the elements will be assembled in the order data%NORDER(1,JDOM), data%NORDER(2, JDOM),..., data%NORDER(data%NELTSB(JDOM), JDOM). Note that if the user has already used MP62 to factorize a matrix with a given sparsity pattern and wishes to factorize another matrix with the same pattern, savings may be made by setting data%ICNTL(9) to a value greater than 1 and leaving data%NORDER unchanged since the factorization of the original matrix.

2.2.2 Output components for data%JOB = 2 or 23

The following component is allocated on each process.

data%INV_LIST is a rank-1 allocatable array of type default INTEGER of size data%NDOM. On exit, data%INV_LIST(JDOM) holds the rank of the process that is to factorize subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%IENT is a rank-1 allocatable array of type default INTEGER of size data%NDOM. On exit, data%IENT(L) holds the number of variable indices for subdomain L (L = 1, 2,..., data%NDOM).

data%ENTRIES is a rank-1 allocatable array of type default INTEGER of size data%NDOM. On exit, data%ENTRIES(L) holds the number of nonzero matrix entries for subdomain L (L = 1, 2,..., data%NDOM). Note that only the entries in the upper triangular part of each element matrix are counted.

data%ICOUNT is a rank-1 allocatable array of type default INTEGER of dimension 0:data%NPROC-1. On exit, data%ICOUNT(IPROC) holds the number of subdomains assigned to process IPROC (IPROC = 0, 1,..., data%NPROC-1).

data%IDOM is a rank-1 allocatable array of type default INTEGER. On exit, on process IPROC, data%IDOM(J) holds the index of the J th subdomain that is to be factorized by the process (J = 1, 2,..., data%ICOUNT(IPROC)).

2.2.3 Input components for data%JOB = 3 or 23

Prior to a call with data%JOB = 3 or 23 the following component must be set on the host (note that if data%JOB = 23, the input components described in Section 2.2.1 must also be set):

data%NRHS is a scalar of type default INTEGER that must be set to the number of right-hand sides. This component is not altered. **Restriction:** data%NRHS ≥ 0.

The remaining components of data that must be set depend on the settings for the control components.

The following component must be set on the host if data%ICNTL(7) $= 0$ (the default).

data%MFRELT is a scalar of type default INTEGER that must be at least as large as the maximum number of variables per element. data%MFRELT determines the record length given in the RECL= specifier of the OPEN statements for the direct-access files for the element data (see data%ICNTL(7) in Section 2.3). This component is not altered.

The following components must be allocated and set on the host if data%ICNTL(7) $= 0$ (the default), 1, or 2.

data%VALNAM is a rank-1 allocatable array of type default CHARACTER*128 that must be allocated with size at least data%NDOM. On entry, data%VALNAM(JDOM) must contain the name of the file holding the element matrices $\mathbf{A}^{(k)}$ belonging to subdomain JDOM (JDOM = 1, 2,..., data%NDOM). data%VALNAM is not accessed if data%ICNTL(7) ≥ 3. This component is not altered.

data%RHSNAM is a rank-1 allocatable array of type default CHARACTER*128 If data%NRHS > 0, data%RHSNAM must be allocated with size at least data%NDOM. On entry, data%RHSNAM(JDOM) must contain the name of the file holding the element right-hand side matrices $\mathbf{B}^{(k)}$ belonging to subdomain JDOM (JDOM = 1, 2,..., data%NDOM). This component is not altered.

The following components must be allocated set on the host only if data%ICNTL(7) = 3 and on each process if data%ICNTL(7) = 4.

data%VALUES is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) that must be allocated and must contain the element matrices $\mathbf{A}^{(k)}$. The element matrices must be in the order given by data%ELTVAR. Only the upper triangular part of each element matrix is required and each element matrix must be stored by columns (in packed triangular form). This component is not altered.

data%RHS is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version). If data%NRHS > 0, data%RHS must be allocated and must contain the element right-hand side matrices $\mathbf{B}^{(k)}$. The $\mathbf{B}^{(k)}$ matrices must be in the order given by data%ELTVAR and each must be held as a full matrix, stored by columns (so that multiple right hand sides are stored as consecutive columns). This component is not altered.

The following component must be allocated and set on each process IPROC if data%ICNTL(7) = 5.

data%RVAL is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) that must be allocated and must contain the element matrix entries $\mathbf{A}^{(k)}$ for the subdomains assigned to process IPROC. The element matrices for subdomain data%IDOM(1) must precede those for data%IDOM(2), and so on, and within each subdomain, the element matrices must be in the order given by data%ELTVAR. Only the upper triangular part of each element matrix is required and each element matrix must be stored by columns (in packed triangular form). The size of data%RVAL on process IPROC must be at least $\sum$data%ENTRIES(L) where the summation is over the subdomains L assigned to IPROC. This component is not altered.

data%RHS_VAL is a rank-1 allocatable array of type default REAL (or doubl precision REAL in the double version) that must be allocated and must contain the element right-hand side matrices $\mathbf{B}^{(k)}$ for the subdomains assigned to process IPROC. The element right-hand side matrices for subdomain data%IDOM(1) must precede those for data%IDOM(2), and so on, and within each subdomain, the element right-hand side matrices must be in the order given by data%ELTVAR and each must be held as a full matrix, stored by columns (so that multiple right hand sides are stored as consecutive columns). The size of data%RHS_VAL on process IPROC must be at least data%NRHS*$\sum$data%IENT(L) where the summation is over the subdomains L assigned to IPROC. This component is not altered.

The following component must be allocated and set if data%ICNTL(13) is nonzero.

data%LENBUF is a rank-2 allocatable array of type default INTEGER that must be allocated with size 2 by data%NDOM+1. It is used to hold the sizes (in words) of the buffers (workspace arrays) used by the frontal solver MA62 for the matrix factors. On entry, data%LENBUF(J, JDOM), J = 1, 2, must hold, respectively, the buffer size for the matrix factor (including the corresponding right-hand sides), and the indices of the variables in the factors on subdomain JDOM (JDOM =1, 2,..., data%NDOM). data%LENBUF(J, data%NDOM+1), J = 1, 2, must hold the corresponding buffer sizes for the interface problem, which is solved on the host. Note that the workspace required by MP62 is dependent on the size of the buffers and for efficiency the buffers should be chosen as large as space permits. If direct-access files are not being used (data%ICNTL(14) = 0), the buffers must be large enough to hold the matrix factors. On exit, data%LENBUF holds the buffer sizes used by MP62. These differ from the input values if the user-supplied values are too large to enable the required workspace to be allocated, or if direct-access files are not being used and one or more of the buffer sizes supplied by the user is too small (see warning +2). **Restriction:** data%LENBUF(:,:) ≥ 1.

The following component must be allocated and set if data%ICNTL(14) < 0.

data%FILES1 is a rank-2 allocatable array of type default CHARACTER*128 that must be allocated with size 2 by data%NDOM+1. On entry, data%FILES1(J, JDOM), J = 1, 2, must hold the names of the direct-access files for the matrix factor and the indices of the variables in the factors on subdomain JDOM (JDOM = 1, 2,..., data%NDOM) and for the interface problem when JDOM = data%NDOM+1. If the user wishes to run further instances of the module before the final call for the first instance (call with JOB = 5), for each instance data%ICNTL(14) must be set to a nonzero value and file names provided by the user in data%FILES1.

The following component must be allocated and set if data%ICNTL(15) < 0.

data%FILES2 is a rank-2 allocatable array of type default CHARACTER*128 that must be allocated with size 2 by data%NDOM. On entry, data%FILES2(1, JDOM)) must hold the name of the sequential file for holding the data that remains in the frontal matrix once the factorization on subdomain JDOM is complete (JDOM = 1, 2,..., data%NDOM). If data%NRHS > 0, data%FILES2(2, JDOM) must hold the name of the sequential file for the corresponding right-hand sides. data%FILES2(2, JDOM) is not accessed if data%NRHS = 0.

2.2.4 Output components for data%JOB = 3 or 23

The following component is allocated if data%NRHS > 0 and is used to hold the solution vector.

data%X is a rank-2 allocatable array of type default REAL (or double precision REAL in the double version) of size data%LARGEST_INDEX (see Section 2.4) by data%NRHS. On exit, if ICNTL(17) = 0, if I has been used to index a variable, then on the host data%X(I, J) holds the solution for variable I to the J-th system and is set to zero otherwise (J = 1, 2,..., data%NRHS). If ICNTL(17) ≠ 0, if the solution for variable I has been computed by the process with rank IPROC, then on process IPROC data%X(I, J) holds the solution for variable I to the J-th system and is set to zero otherwise (IPROC = 0, 1,..., data%NPROC).

2.2.5 Input components for data%JOB = 4

Prior to a call with data%JOB = 4 the following components must be set on the host:

data%NRHS is a scalar of type default INTEGER that must be set to the number of right-hand sides. This component is not altered. **Restriction:** data%NRHS ≥ 1.

data%X is a rank-2 allocatable array of type default REAL (or double precision REAL in the double version) that must be allocated with size data%LARGEST_INDEX (see Section 2.4) by data%NRHS. On entry, data%X must be set by the user so that if I is used to index a variable, data%X(I, J) is the corresponding component of the right-hand side for the J-th system (J = 1, 2,..., data%NRHS). This component is altered.

2.2.6 Output components for data%JOB = 4

The following component is used to hold the solution vector.

data%X is a rank-2 allocatable array of type default REAL (or double precision REAL in the double version) of size data%LARGEST_INDEX (see Section 2.4) by data%NRHS. On exit, if ICNTL(17) = 0, if I has been used to index a variable, then on the host data%X(I, J) holds the solution for variable I to the J-th system and unchanged otherwise (J = 1, 2,..., data%NRHS). If ICNTL(17) ≠ 0, if the solution for variable I has been computed by the process with rank IPROC, then on process IPROC, data%X(I, J) holds the solution for variable I to the J-th system (IPROC = 0, 1,..., data%NPROC).

2.3 Control components

On exit from the initial call (data%JOB = 1), the control components of data are set to default values. If the user wishes to use values other than the defaults, the corresponding components of data should be reset on the host process after the initial call and prior to a call with data%JOB = 2 or 23.

data%ICNTL is a rank-1 array of type default INTEGER and size 30.

ICNTL(1) is the stream number for error messages and has the default value 6. Printing of error messages is suppressed if ICNTL(1) < 0.

ICNTL(2) is the stream number for warning messages and has the default value 6. Printing of warning messages is suppressed if ICNTL(2) < 0.

ICNTL(3) is the stream number for diagnostic printing and has the default value 6. Printing is suppressed if ICNTL(3) < 0.

ICNTL(4) is used to control printing of diagnostic messages (all printing is on the host only). It has default value 1. Possible values are:

- ≤0 No printing.
- 1 Error and warning messages only.
- 2 As 1, plus some additional diagnostic printing.
- 3 As 2, but timings of parts of the code (elapsed wall clock times in seconds) are also printed.

ICNTL(5) is the block size for the numerical factorization of the frontal matrix. It controls the trade-off between Level 2 and Level 3 BLAS. If ICNTL(5) = 1, Level 2 BLAS is used to form the Schur complement. If ICNTL(5) ≥ 1, the Level 3 BLAS routine _GEMM is used with internal dimension ICNTL(5). Increasing ICNTL(5) increases the number of flops since symmetry is not exploited as well. The optimal value for ICNTL(5) depends on the computer being used. A value of ICNTL(5) less than one is treated as one and, if at some stage of the factorization, ICNTL(5) has a value which is larger than the current front size, ICNTL(5) is treated as the front size. Typical range: 16 to 64. Default value: 16.

ICNTL(6) controls whether zeros in the front are exploited. Zeros within the front are ignored if ICNTL(6) = 0. The default value is 1.

ICNTL(7) is used to control how the user wishes to supply the element matrices $\mathbf{A}^{(k)}$ and the element right-hand side matrices $\mathbf{B}^{(k)}$ on a call with data%JOB = 3 or 23. There are 4 options:

- 0 The element data and element right-hand sides are held in direct-access files. The data required by the process with rank IPROC must be readable by that process (IPROC = 0, 1, ..., data%NPROC-1). For each subdomain, the element data is read in element-by-element as required by the corresponding process. This minimises storage requirements and data movement between processes. After a call with data%JOB = 2, data%INV_LIST(JDOM) holds the rank of the process that is to factorize subdomain JDOM. For each subdomain JDOM to be factorized by process IPROC, there must be an unformatted direct-access file holding the values of the entries in the element matrices and, if data%NRHS > 0, another holding the values of the entries in the element right-hand side matrices, that can be read by process IPROC. The record length given in the RECL= specifier of the OPEN statements for the direct-access files holding the element matrices and element right-hand side matrices must be that required for REAL (or double precision REAL in the double version) arrays of size data%MFRELT*data%MFRELT, and data%MFRELT*data%NRHS, respectively. The element matrices and element right-hand side matrices must be written to the direct-access files in the same order as they are held in data%ELTVAR. Only the upper triangular part of each element matrix is required and each element matrix must be stored by columns (in packed triangular form). Each element right-hand side matrix must be held as a full matrix, stored by columns (so that multiple right hand sides are stored as consecutive columns). The names of the files for subdomain JDOM must be given in data%VALNAM(JDOM) and data%RHSNAM(JDOM), respectively (JDOM = 1, 2, ..., data%NDOM).
- 1 As 0, except the element data and element right hand sides are held in unformatted sequential files. In this case, all the element data for a subdomain is read in by the process to which it is assigned at once. This requires more memory.

2 As 1, except the host must be able to read all the files. For each subdomain, the data is read by the host and then passed to the process assigned to that subdomain before the factorization begins. This requires the host to have more memory.

3 The user must supply the element data and the element right-hand sides in memory on the host using data%VALUES and data%RHSVAL. This option avoids reading from direct-access or sequential files but it does involve more data movement between processes than ICNTL(7) = 0 or 1.

4 The user must supply the element data and the element right-hand sides in memory on each process using data%VALUES and data%RHSVAL. Supplying the data in this way avoids reading from direct-access or sequential files as well as data movement between processes. This option is suitable for shared memory machines.

5 On each process, the user must supply the element data and the element right-hand sides for each of the subdomains assigned to it in memory using data%RVAL and data%RHS_VAL. Supplying the data in this way avoids reading from direct-access or sequential files as well as data movement between processes; the amount of data required to be held on each process is less than for ICNTL(7) = 4 (assuming more than one process is used).

Restriction: ICNTL(7) = 0, 1, 2, 3, 4, or 5.

ICNTL(8) is not currently used. It is given the default value 0.

ICNTL(9) controls the order in which the elements are assembled in the subdomains. If ICNTL(9) = 0 (the default), the assembly order is generated automatically using MC53 during the analyse phase (data%JOB = 2). If ICNTL(9) > 0, the user must specify the assembly order using the array data%NORDER. Otherwise, the elements are assembled in the order in which they are given in data%ELTVAR.

ICNTL(10) is used to control whether the user wishes to decide which process is to factorize which subdomain. If ICNTL(10) = 0 (the default), this choice is made automatically during the analyse phase (data%JOB = 2). Otherwise, the user must choose a process for each subdomain using data%INV_LIST.

ICNTL(11) controls whether a packed triangular form is used by MA62. If ICNTL(11) = 0 (the default value), a packed triangular form is used. In this case, the Level 2 BLAS routine _TPSV is used to perform triangular solves. If ICNTL(11) is reset to a nonzero value, a packed triangular form is not used. This increases the real storage required for the factors but the advantage is that, for multiple right-hand sides, the Level 3 BLAS routine _TRSM is used.

ICNTL(12) controls whether the user wants the direct-access files used to hold the matrix factors to be deleted at the end of the computation. If ICNTL(12) = 0 (the default), when the final call is made to MP62 (data%JOB = 5), the direct-access files are deleted. Otherwise, the direct-access files are disconnected but not deleted. ICNTL(12) is not used if ICNTL(14) is equal to 0.

ICNTL(13) controls the size of the buffers (work arrays) associated with the direct-access files used to hold the matrix factors. If ICNTL(13) = 0 (the default), the buffer sizes are chosen by the code. If ICNTL(13) is nonzero, the user must set buffer sizes in data%LENBUF.

ICNTL(14) controls whether or not direct-access files are used to hold the matrix factors. If ICNTL(14) = 0 (the default), direct-access files are **not** used and the factors are held main memory. Otherwise, unformatted direct-access files are used. If ICNTL(14) > 0, the code automatically names the files and they are written to the current directory. The files for the factor on subdomain 1 are called factor.0001, integ.0001, on subdomain 2 they are factor.0002, integ.0002, and so on. The files for the factor for the interface problem are factor_interf, integ_interf. If ICNTL(14) < 0, the user must supply names for the files in data%FILES1. If the user wishes to run a second instance of the module before the final call for the first instance (data%JOB = 5), data%ICNTL(14) **must** be set to a negative value and file names provided by the user in data%FILES1.

ICNTL(15) controls whether or not sequential files are used to hold the data that remains in the frontal matrix and

corresponding right-hand side matrix once the factorization on the subdomain is complete. If ICNTL(15) = 0 (the default), files are **not** used. Otherwise, unformatted sequential files are used (using files reduces storage requirements but the extra I/O involved can increase the overall computational time). If ICNTL(15) > 0, the code automatically names the files and they are written to the current directory. The remaining data for subdomain 1 is written to files fvar.0001 and (if data%NRHS > 0) frhs.0001, for subdomain 2 the files are fvar.0002 and frhs.0002, and so on. If ICNTL(15) < 0, the user must supply names for the files for the factors in data%FILES2. If the user wishes to run a second instance of the module before the final call for the first instance (call with JOB = 5), data%ICNTL(15) **must** be set to a negative value and file names provided by the user in data%FILES2.

ICNTL(16) has the default value 16. ICNTL(16) is the minimum number of variables that are eliminated at each stage of the factorization. Increasing ICNTL(16) 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.

ICNTL(17) controls whether or not the solution vector is assembled on the host. If ICNTL(17) = 0 (the default), the solution is assembled in data%X on the host. Otherwise, on exit, each process holds the part of the solution vector that it computed.

ICNTL(18) controls whether or not the BLAS are used during the solve phase JOB = 4. If ICNTL(18) = 1 and data%NRHS = 1, BLAS are not used. Otherwise, BLAS are used. The default value is 0.

ICNTL(19) to ICNTL(30) are not currently used but are set to zero.

data%CNTL is a rank-1 array of type default REAL (or double precision REAL in the double version) and size 10.

CNTL(1) has the default value zero. If, during the factorization, the absolute value of any pivot is less than or equal to CNTL(1), the computation terminates and the matrix is declared to be not positive definite (see error –12).

CNTL(2) to CNTL(10) are not currently used but are set to zero.

2.4 Information components

The components of data described in this section are used to hold information that may be of interest to the user. Some of the information is available on each process and some only on the host.

2.4.1 Information on each process

The following information is significant on each process. The information is available after a call to MP62 with data%JOB = 1, 2, 3, 23, 4, or 5.

data%ERROR is a variable of type default INTEGER that is used as an error and a warning flag. A nonzero value indicates an error has been detected or a warning has been issued (see Section 2.5). If an error is detected, the information contained in the other components of data described in this section may be incomplete.

data%NPROC is a variable of type default INTEGER that holds the number of processes used by MP62. data%NPROC is the number of processes associated with the communicator data%COMM and is set by a call within MP62 to MPI_COMM_SIZE.

data%RANK is a variable of type default INTEGER that holds the rank of the process in the global communicator data%COMM. The host is defined to be the process with data%RANK = 0.

2.4.2 Information available on the host

The following information is significant **only** on the host. If an error is detected (see Section 2.5), the information may be incomplete.

Information available on exit from a call with data%JOB = 2 **or** 23**.**

data%LARGEST_INDEX is a variable of type default INTEGER that holds the largest integer used to index a variable in the finite element domain.

data%MAX_NDF is a variable of type default INTEGER that holds the maximum number of variables per element.

data%NGUARD is a rank-1 allocatable array of type default INTEGER of size data%NDOM. data%NGUARD(JDOM) holds the number of interface variables for subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%FLAG_72 is a rank-1 allocatable array of type default INTEGER of size data%NDOM. data%FLAG_72(JDOM) holds the MA72A/AD error flag for subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%NDF is a rank-1 allocatable array of type default INTEGER of size data%NDOM. data%NDF(JDOM) holds the number of variables in subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%NFRONT is a rank-1 allocatable array of type default INTEGER of size data%NDOM. data%NFRONT(JDOM) holds the maximum frontsizes for the frontal elimination on subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%RMS is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) of size data%NDOM. data%RMS(JDOM) holds the root mean squared frontsize for the frontal elimination on subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%OPS is a rank-1 allocatable array of type default REAL (or double precision REAL in the double version) of size data%NDOM. data%OPS(JDOM) holds an estimate of the number of floating-point operations (flops) in the innermost loops for the frontal elimination on subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%FLSIZE is a rank-2 allocatable array of type default REAL of size 2 by data%NDOM. data%FLSIZE(J, JDOM), J = 1, 2 hold, respectively, the estimated storage needed for the real and integer data for the matrix factor on subdomain JDOM (JDOM = 1, 2,..., data%NDOM).

data%NORDER is a rank-2 allocatable array of type default INTEGER of size $\max_{1 \leq \text{JDOM} \leq \text{data\%NDOM}}$ data%NELTSB(JDOM)+1 by data%NDOM. Within subdomain JDOM, the elements are locally labelled 1, 2,..., data%NELTSB(JDOM), according to the order in which the user inputs the variable lists in data%ELTVAR. On subdomain JDOM, the elements will be assembled during the factorize phase in the order data%NORDER(1,JDOM), data%NORDER(2,JDOM),..., data%NORDER(data%NELTSB(JDOM),JDOM).

Information available on exit from a call with data%JOB = 3 **or** 23**.**

data%STATIC is a variable of type default INTEGER that holds the total number of static condensations performed.

data%SINGULAR is a variable of type default LOGICAL that is set to .TRUE. if the matrix is found to be singular.

data%FLOPS is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of floating-point operations (flops) in the innermost loops of the factorization (this is the total for all the subdomains and the interface).

data%STORREAL is a variable of type default REAL (or double precision REAL in the double version) that holds the total number of entries in the factors.

data%STORINT is a variable of type default REAL (or double precision REAL in the double version) that holds the total storage for the indices of the variables in the factors in INTEGER words.

data%NLEFT is a rank-1 allocatable array of type default INTEGER of size data%NBLOCK. data%NLEFT(JDOM) holds the number of variables left in the front at the end of the elimination process for subdomain JDOM (JDOM = 1, 2,..., data%NDOM). Note that, if ICNTL(16) > 1, data%NLEFT(JDOM) may be larger than data%NGUARD(JDOM).

data%INFO_62 is a rank-2 allocatable array of type default INTEGER of size 20 by data%NDOM+1. data%INFO_42(1:20, JDOM) holds the MA62B/BD integer information array for subdomain JDOM (JDOM = 1, 2,..., data%NDOM) and for JDOM = data%NDOM+1, it holds the MA62B/BD integer information array for the interface problem. Note that on the subdomains the information in data%INFO_62 is incomplete since the factorization on the subdomain is a partial factorization (interface variables not eliminated).

data%RINFO_62 is a rank-2 allocatable array of type default REAL (or dou precision REAL in the double version) of size 20 by data%NDOM+1. data%RINFO_42(:, JDOM) holds the MA62B/BD real information array for subdomain JDOM (JDOM = 1, 2,..., data%NDOM) and for JDOM = data%NDOM+1 it holds the MA62B/BD real information array for the interface problem. Note that on the subdomains the information in data%RINFO_62 is incomplete since the factorization on the subdomain is a partial factorization (interface variables not eliminated).

data%INFO_63 is a rank-1 array of type default INTEGER of size 15. data%INFO_63 holds the MC63A/AD integer information array for the interface problem.

data%RINFO_63 is a rank-1 array of type default REAL (or double precision REAL in the double version) of size 6. data%RINFO_63 holds the MC63A/AD real information array for the interface problem.

2.5 Error diagnostics

On successful completion, a call to MP62 will exit with the component data%ERROR set to 0. Other values for data%ERROR and the reasons for them are given below.

2.5.1 Error diagnostics for data%JOB = 1

–1 MPI has not been initialized by the user. Immediate return. An error message is printed on the default output stream.

2.5.2 Error and warning diagnostics for data%JOB = 2 or 23

A negative value for data%ERROR is associated with a fatal error. Error messages are output on stream data%ICNTL(1). Possible negative values are:

–2 Either data%NDOM ≤ 1 or data%NDOM > data%NELT.

–4 Either data%ELTVAR is not allocated or has been allocated with size less than data%ELTPTR(data%NELT+1)-1. This error is also returned if one or more variable indices in data%ELTVAR is out of range (ie. is less than 1) or there are duplicate entries in a row. data%IOUT and data%IDUP hold, respectively, the number of out of range and duplicate entries.

–5 Error detected in data%ELTPTR. Either data%ELTPTR has not been allocated or has been allocated with size less than data%NELT+1, or the entries of data%ELTPTR are not monotonic increasing.

–6 data%ICNTL(7) out of range (ie. data%ICNTL(7) ≠ 0, 1, 2, 3, 4, or 5).

–7 Either the array data%NELTSB has not been allocated or has been allocated with size less than data%NDOM, or data%NELTSB(JDOM) < 1 for one or more subdomain JDOM (1 ≤ JDOM ≤ data%NDOM).

–8 One or more subdomain have no interface variables.

–9 Either the array data%INV_LIST has not been allocated or has been allocated with size less than data%NBLOCK, or an entry in data%INV_LIST is out of range (data%ICNTL(10) nonzero).

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT. If the user is not using direct-access files (data%ICNTL(14) = 0), it may be possible to avoid this error by rerunning with data%ICNTL(14) ≠ 0, or if data%ICNTL(13) is nonzero, by reducing the buffer sizes held in data%LENBUF.

–13 data%JOB does not have the same value on all processes or has an invalid value.

–18 The call follows a call with data%JOB = 5.

–24 Either the array data%NORDER has not been allocated or has been allocated with an incorrect size (ICNTL(9) > 0). This error is also returned if the supplied element order is found not to be a permutation for one or more of the subdomains.

Warning messages are associated with positive values of data%ERROR. Warning messages are output on

data%ICNTL(2). Possible warnings are:

+1 Warning issued by MC53A/AD for one or more subdomain.

2.5.3 Error diagnostics for data%JOB = 3 or 23

A negative value for data%ERROR is associated with a fatal error. Error messages are output on stream data%ICNTL(1). Possible values are:

–10 If data%ICNTL(7) = 0, 1, or 2, either data%VALNAM or data%RHSNAM is not allocated or is allocated with size less than data%NDOM. If data%ICNTL(7) = 3 or 4, either data%VALUES or data%RHS is not allocated or is allocated with incorrect size. If data%ICNTL(7) = 5, either data%RVAL or data%RHS_VAL is not allocated or is allocated with incorrect size.

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT.

–12 Attempt to use a pivot of absolute value less than or equal to CNTL(1).

–13 data%JOB does not have the same value on all processes or has an invalid value.

–14 Error in Fortran INQUIRE statement. data%IOSTAT holds the IOSTAT parameter.

–15 Error when writing to a direct-access file.

–16 Error when reading from a direct-access file.

–17 Error in Fortran OPEN statement.

–19 An error was returned on a previous call or the call follows a call with data%JOB = 1 (no data%JOB = 2 call) or follows a call with data%JOB = 5.

–20 Failed to find a unit to which a file could be connected.

–21 Either data%LENBUF has not been allocated or has been allocated with incorrect size, or data%LENBUF(1, JDOM) ≤ 0, or data%LENBUF(2, JDOM) ≤ 0 (1 ≤ JDOM ≤ data%NDOM+1) (data%ICNTL(13) nonzero).

–22 data%NRHS < 0.

–23 data%ICNTL(7) = 0 and data%MFRELT is less than the maximum number of variables per element. data%MAX_NDF holds the maximum number of variables per element.

–25 data%FILES1 is either not allocated or is allocated but with incorrect size (ICNTL(14) < 0).

–26 data%FILES2 is either not allocated or is allocated but with incorrect size (ICNTL(15) < 0).

Warning messages are associated with positive values of data%ERROR. Warning messages are output on data%ICNTL(2). Possible warnings are:

+2 One or more of the user-supplied buffer sizes in data%LENBUF have been altered (data%ICNTL(13) nonzero).

+3 Combination of the above warnings.

2.5.4 Error diagnostics for data%JOB = 4

A negative value for data%ERROR is associated with a fatal error. Error messages are output on stream data%ICNTL(1). Possible values are:

–11 Error in Fortran ALLOCATE statement. The STAT parameter is returned in data%STAT.

–13 data%JOB does not have the same value on all processes or has an invalid value.

–16 Error when reading from a direct-access file.

–17 Error in Fortran OPEN statement.

–19 An error was returned on a previous call or the call was not preceded by a call with data%JOB = 3 or 23, or follows a call with data%JOB = 5.

–22 data%NRHS < 1. This error is also returned if data%NRHS ≥ 1 but data%X is either not allocated or is allocated but with incorrect size.

2.5.5 Error diagnostics for data%JOB = 5

A negative value for data%ERROR is associated with a fatal error. Error messages are output on stream data%ICNTL(1). Possible values are:

–13 data%JOB does not have the same value on all processes or has an invalid value.

3 GENERAL INFORMATION

3.1 Summary of information.

Use of common: Common blocks are not used.

Other routines called directly: The HSL routines HSL_MP01, MA62I/ID, MA62A/AD, MA62B/BD, MA62C/CD, MA62J/JD, MA62P/PD, MA72A/AD, MA72B/BD, MA72C/CD, MC53I/ID, MC53A/AD, MC63I/ID, MC63A/AD, KB08A/AD. Subroutines private to the module are MP62B/BD, MP62C/CD, MP62L/LD, MP62M/MD and MP62N/ND. In addition, MPI routines are called.

Workspace: Workspace is allocated by the code as required. The amount of workspace needed is dependent upon how the element matrices are stored, on data%LENBUF, and on the largest integer used to index a variable.

Input/output: The output streams for the error and warning messages are data%ICNTL(1) and data%ICNTL(2) (see Section 2.3). The output stream for diagnostic printing is data%ICNTL(3).

Restrictions:
data%NDOM > 1,
data%NELT ≥ data%NDOM,
data%NRHS ≥ 0 (data%JOB = 3 or 23),
data%NRHS ≥ 1 (data%JOB = 4),
data%NELTSB(:) > 0,
0 ≤ data%INV_LIST(:) < data%NPROC (data%ICNTL(10) nonzero),
data%LENBUF(:,:) ≥ 1 (data%ICNTL(13) nonzero).

Portability: Fortran 95 + TR 15581 (allocatable components) with MPI for message passing.

Changes between Version 1.0.0 and Version 2.0.0:
The addition of HSL_MP01 to HSL has allowed the source form to be changed to free format and means that the user of no longer needs an INCLUDE line for the MPI constants. All the pointer array components have been changed to allocatable components, which should be more efficient and avoids any danger of memory leakage.

4 METHOD

HSL_MP62 implements a multiple front algorithm. The code is a symmetric positive definite version of HSL_MP42. Details of the algorithm are given in Duff and Scott (1994) and Scott (1999).

data%JOB = 1

The control components are given default values.

data%JOB = 2

The input data is first checked for errors. The control components and scalar input components are then broadcast from the host to all processes. The host process calls MA72A/AD to generate lists of interface variables (the guard elements). Unless data%ICNTL(10) has been reset to a nonzero value, the host assigns each subdomain to a process. This division of the subdomains between the processes aims to balance the floating-point operations.

data%JOB = 3

Data for each subdomain is sent from the host to its assigned process IPROC. For each of its assigned subdomains, IPROC calls MC53 to order the elements. Process IPROC generates unit numbers for the direct-access files that will hold the matrix factors, calls the analyse and factorize phases of MA62, and uses MA72B/BD to preserve the partial factorization.

Data from MA72B/BD is sent to the host. The host reorders the subdomains using MC63 and uses MA62 to solve the interface problem. If data%NRHS > 0, the solution for the interface problem is sent to each process, and on each process MA72C/CD is called to perform the back substitution for its assigned subdomains.

data%JOB = 4

The host first performs error checks and broadcasts the number of right-hand sides to each process. For each of its assigned subdomains, process IPROC calls MA72C/CD to perform forward substitution. The partial solution vectors are sent to the host. Forward and back substitution for the interface problem is performed by the host using MA62C/CD. The solution for the interface problem is sent to each process, and on each process MA72C/CD is called to perform the back substitution on its subdomains.

data%JOB = 5

Arrays allocated by the code are deallocated, the direct-access files used to hold the matrix factors are closed and (optionally) are deleted.

MPI is used throughout for message passing.

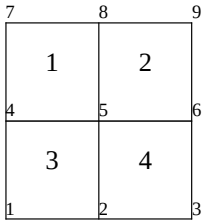
References

Duff, I. S. and Scott, J. A. (1994). The use of multiple fronts in Gaussian elimination. Rutherford Appleton Laboratory Report RAL-94-040.

Scott, J. A. (1999). The design of a parallel frontal solver. Rutherford Appleton Laboratory Report RAL-TR-99-075.

5 EXAMPLE OF USE

We wish to solve 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, $1 \leq i \leq 6$ (the nodes 7, 8, and 9 are assumed constrained). The mesh is divided into 2 subdomains in which elements 1 and 2 comprise subdomain 1 and elements 3 and 4 comprise subdomain 2. We supply one right-hand side with the elements and then use data%JOB = 4 to solve for a further two right-hand sides.



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

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

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

$$\begin{pmatrix} 3. \\ 8. \end{pmatrix} \qquad \begin{pmatrix} 5. \\ 10. \end{pmatrix} \qquad \begin{pmatrix} 12. \\ 9. \\ 12. \\ 11. \end{pmatrix} \qquad \begin{pmatrix} 14. \\ 8. \\ 17. \\ 14. \end{pmatrix}.$$

The (assembled) right-hand sides that are used in the call with `data%JOB` = 4 are

$$\begin{pmatrix} 6 \\ 1 \\ 2 \\ 7 \\ 4 \\ -1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 12 \\ 4 \\ 2 \\ 4 \\ 9 \\ 11 \end{pmatrix}.$$

The following program is used to solve this problem.

```
! Code to run MP62 on a finite element mesh composed of
! two subdomains.

      USE HSL_MP62_DOUBLE
      IMPLICIT NONE

      TYPE (MP62_DATA) data

      INTEGER ERCODE,NE,RHSCRD,VALCRD

! Start MPI
      CALL MPI_INIT(ERCODE)

! Define a communicator for the package
      data%COMM = MPI_COMM_WORLD

! Initialize instance
      data%JOB = 1
      CALL MP62AD(data)
! For this simple example, no need to order the elements
! within each subdomain. Reset data%ICNTL(9).
      data%ICNTL(9) = -1
! We will read in all matrix values to data%VALUES and data%RHSVAL
      data%ICNTL(7) = 3

! Read in data on host
      IF (data%RANK .EQ. 0) THEN
```

```
! Number of subdomains is 2
        data%NDOM = 2
! Solve for 1 right hand side
        data%NRHS = 1

! Read in the number of elements in whole domain
        OPEN (UNIT=5,FILE='mp62ads.data')
        READ (5,FMT=*) data%NELT

! Read number of elements in each subdomain
        ALLOCATE(data%NELTSB(1:data%NDOM))

        READ (5,FMT=*) data%NELTSB(1:data%NDOM)
! Read element variable lists
        ALLOCATE(data%ELTPTR(1:data%NELT+1))
        READ (5,FMT=*) data%ELTPTR(1:data%NELT+1)

        NE = data%ELTPTR(data%NELT+1) - 1
        ALLOCATE(data%ELTVAR(1:NE))
        READ (5,FMT=*) data%ELTVAR(1:NE)

! Read in reals
        READ (5,FMT=*) VALCRD
        ALLOCATE(data%VALUES(1:VALCRD))
        READ (5,FMT=*) data%VALUES(1:VALCRD)

! Read in element right hand sides
        READ (5,FMT=*) RHSCRD
        ALLOCATE(data%RHS(1:RHSCRD))
        READ (5,FMT=*) data%RHS(1:RHSCRD)

      END IF
      CALL MPI_BARRIER(data%COMM,ERCODE)
! Analyse and factorize phases
      data%JOB = 23
      CALL MP62AD(data)
      IF (data%ERROR.LT.0) GO TO 20
      IF (data%RANK.EQ.0)   THEN
         WRITE (*,FMT=9000)
         WRITE (*,FMT=9010) data%X(1:data%LARGEST_INDEX,1)
      END IF

! We want to solve for further right-hand sides.
! Read assembled right-hand side in X.
      IF (data%RANK.EQ.0)   THEN
         WRITE (*,FMT=9020)
         data%NRHS = 2
         DEALLOCATE(data%X)
         ALLOCATE(data%X(1:data%LARGEST_INDEX,1: data%NRHS))
         READ (5,FMT=*) data%X(1:data%LARGEST_INDEX,1)
         READ (5,FMT=*) data%X(1:data%LARGEST_INDEX,2)
      END IF

      data%JOB = 4
      CALL MP62AD(data)
      IF (data%ERROR.LT.0) GO TO 20

      IF (data%RANK.EQ.0) THEN
         WRITE (*,FMT=9000)
         WRITE (*,FMT=9010) data%X(1:data%LARGEST_INDEX,1)
         WRITE (*,FMT=9010) data%X(1:data%LARGEST_INDEX,2)
```

```
      END IF

! Final call
  20  data%JOB = 5
      CALL MP62AD(data)
      CALL MPI_FINALIZE(ERCODE)

 9000 FORMAT (/' The solution is:')
 9010 FORMAT (6F12.4)
 9020 FORMAT (/' Now solving for further right-hand sides.')
      STOP
      END
```

The input data used for this problem is:

```
    4
    2   2
    1   3   5   9  13
    4   5   5   6   4   5   1   2   5   6   2   3
   26
    2.   1.   7.   3.   2.   8.   4.   3.   1.   2.   3.   6.
    3.   2.   1.   5.   2.   1.   3.   8.   2.   2.   3.   2.
    5.   4.
   12
    3.   8.   5.  10.  12.   9.  12.  11.  14.   8.  17.  14.
    6.   1.   2.   7.   4.  -1.
   12.   4.   2.   4.   9.  11.
```

When run using 1 or 2 processes, this produces the following output:

```
 The solution is:
      1.0000      1.0000      1.0000      1.0000      1.0000      1.0000

 Now solving for further right-hand sides.

 The solution is:
      1.0000      1.0000      0.0000      1.0000     -1.0000      0.0000
      2.0000      0.0000      0.0000      0.0000      0.0000      1.0000
```