1 SUMMARY

HSL_MA85 uses a direct method to solve **large-scale diagonally-weighted linear least squares problems**. Given an $m \times n$ ($m \geq n$) matrix $A = \{a_{ij}\}$, an $m \times m$ diagonal matrix of weights W, and an $m-$vector b, HSL_MA85 solves either the least squares problem

$$\min_x \|W(Ax-b)\|_2^2, \tag{1.1}$$

or the regularized least squares problem

$$\min_x \|W(Ax-b)\|_2^2 + \alpha\|x\|_2^2, \tag{1.2}$$

where $\alpha > 0$ is a regularization parameter chosen by the user. The matrix A may contain one or more rows that are to be treated as dense but must otherwise be sparse. Rows of A that lead to a large amount of fill in the normal matrix (see (1.3) below) should be treated as dense (they may contain fewer than n non zero entries but generally have more non zeroes than the other rows of A). The package offers the option of (i) a Cholesky-based approach or (ii) an approach that uses a symmetric indefinite solver applied to the (regularized) augmented system.

Approach (i) (Cholesky)
If there are no rows to be treated as dense, the sparse Cholesky package HSL_MA87 is used to solve the $n \times n$ positive definite normal equations

$$Cx = A^T W^2 b, \qquad C = A^T W^2 A + \alpha I. \tag{1.3}$$

If A has $m_d \geq 1$ dense rows, and assuming these rows have been permuted to be the last rows of A, the problem becomes

$$\min_x \left\| \begin{pmatrix} W_s A_s \\ W_d A_d \end{pmatrix} x - \begin{pmatrix} W_s b_s \\ W_d b_d \end{pmatrix} \right\|_2, \tag{1.4}$$

where $A_s \in R^{m_s \times n}$ is sparse, $A_d \in R^{m_d \times n}$ is dense, $W_s \in R^{m_s \times m_s}$ and $W_d \in R^{m_d \times m_d}$ are diagonal, and $b_s \in R^{m_s}$, $b_d \in R^{m_d}$, with $m = m_s + m_d$, $m_s \gg m_d$. The solution is obtained using a block signed Cholesky factorization of the equivalent $(n+m_d) \times (n+m_d)$ system

$$\begin{pmatrix} C_s & A_d^T W_d \\ W_d A_d & -I \end{pmatrix} \begin{pmatrix} x \\ W_d A_d x \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}, \tag{1.5}$$

where $C_s = A_s^T W_s^2 A_s + \alpha I$ and $c = A_s^T W_s^2 b_s + A_d^T W_d^2 b_d$. If $\alpha = 0$, iterative refinement may be performed; otherwise the factors may be used as a preconditioner for the iterative solver CGLS to recover the solution of the unregularized problem (1.1).

Approach (ii) (symmetric indefinite)
If there are no rows to be treated as dense, the $(n+m) \times (n+m)$ augmented system

$$Ky = c, \tag{1.6}$$

where

$$K = \begin{pmatrix} \beta I & WA \\ A^T W & -\alpha I \end{pmatrix}, \qquad y = \begin{pmatrix} \beta^{-1} r \\ x \end{pmatrix}, \qquad c = \begin{pmatrix} Wb \\ 0 \end{pmatrix}, \qquad r = W(Ax-b), \tag{1.7}$$

is solved using the sparse symmetric indefinite solver HSL_MA97, with threshold partial pivoting. This computes a factorization $PKP^T = LDL^T$, where P is a permutation, L is unit lower triangular and D is block diagonal, with blocks of size 1 and 2. Here $\beta > 0.0$ is a scaling parameter that may be selected by the user.

If A has $m_d \geq 1$ dense rows, and assuming as in (1.4) these are the last rows of A, HSL_MA97 is used to solve the $(n+m_d)\times(n+m_d)$ reduced augmented system

$$K_r \begin{pmatrix} x \\ \beta^{-1} r_d \end{pmatrix} = \begin{pmatrix} -\beta^{-1} A_s^T W_s^2 b_s \\ W_d b_d \end{pmatrix}, \qquad K_r = \begin{pmatrix} -C_s & A_d^T W_d \\ W_d A_d & \beta I \end{pmatrix}, \qquad b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \qquad r = \begin{pmatrix} r_s \\ r_d \end{pmatrix}, \tag{1.8}$$

where $C_s = \beta^{-1} A_s^T W_s^2 A_s + \alpha I$. If $\alpha = 0$, GMRES may be used to refine the solution and if $\alpha > 0$, GMRES may be used to recover the solution of the unregularized problem (1.2).

ATTRIBUTES — Version: 1.0.0 (June 2020). **Interfaces:** Fortran, MATLAB. **Types:** Real (single, double). **Calls:** MI24, HSL_MA87, HSL_MA97, HSL_MC68, and (optionally) METIS_NODEND. Lapack routines _gesv, _syrk, _potrf, and _potrs. BLAS routines _dot, _nrm2, _axpy, _gemv, _gemm. **Language:** Fortran 2003. **Date:** June 2020. **Origin:** J. A. Scott, STFC Rutherford Appleton Laboratory. **Parallelism:** Uses OpenMP and its runtime library. **Remark:** The development of this package was supported by EPSRC grant EP/M025179/1.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement
Single precision version

```
use hsl_MA85_single
```

Double precision version

```
use hsl_MA85_double
```

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

The following procedures are available to the user:

(a) MA85_check_matrix takes the matrix A and (optionally) the weights W and checks for out-of-range entries and duplicates. In addition, it checks for zero weights and removes null rows and columns of WA. If b is entered, entries corresponding to null rows of WA are removed. Additionally, the routine may identify rows that are considered dense and it permutes such rows to be the last rows of the matrix.

(b) MA85_checkb may be called following a call to MA85_check_matrix. It takes b and/or W and cleans them, that is, removes zero weights from W and removes entries of b that correspond to null rows in WA. It also permutes the entries of b and W consistently with the row permutation performed by MA85_check_matrix.

(c) MA85_factor computes the factorization. The user selects Approach (i) or Approach (ii).

(d) MA85_solve uses the factors generated by MA85_factor to solve the least squares problem (1.1) and to compute the weighted residual vector $r = W(Ax-b)$. More than one call to MA85_solve with different (cleaned) b may follow a call to MA85_factor.

(e) MA85_finalise should be called after all calls to MA85_solve are complete.

(f) MA85_LS provides a simple interface that performs the matrix checking, factor, solve and finalise in a single call.

2.2 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types MA85_keep, MA85_control and MA85_info. The following pseudo code illustrates this.

```
use hsl_MA85_double
...
type (MA85_keep) :: keep
type (MA85_control) :: control
type (MA85_info) :: info
...
```

The components of MA85_control and MA85_info are explained in Sections 2.6 and 2.7. The components of MA85_keep are used to pass data between the subroutines of the package and must not be altered by the user.

2.3 OpenMP

OpenMP is used by the HSL_MA85 package to provide parallelism for shared memory environments. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp). The number of threads may be controlled at runtime by setting the environment variable OMP_NUM_THREADS.

2.4 METIS

The HSL_MA85 package optionally uses the METIS graph partitioning library available from the University of Minnesota website. If METIS is not available, the user must link with the supplied dummy subroutine METIS_NodeND. In this case, the METIS ordering option will not be available to the user and, if selected, MA85_factorize will return with an error.

Important: At present, HSL_MA85 only supports METIS version 4, not the latest version 5 releases.

2.5 Argument lists and calling sequences

2.5.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments. In each call, optional arguments follow the argument info. **Optional arguments should be called by keyword, not by position**.

2.5.2 Integer and package types

INTEGER denotes default INTEGER and INTEGER(long) denotes INTEGER(kind=selected_int_kind(18)). We use the term **package type** to mean default real if the single precision version is being used and double precision real for the double precision version.

2.5.3 To check and clean the matrix data and optionally permute dense rows to the end

It is recommended that MA85_check_matrix is called to check the input matrix data; if this is not done, the behaviour of MA85_factor and MA85_solve is not guaranteed (these routines perform **no checks** on the input data). If the user wants to use MA85_LS, calling MA85_check_matrix is unnecessary because it is called internally by MA85_LS.

MA85_check_matrix checks the input matrix A for duplicate entries and out-of-range entries (any such entries result in an error), checks for explicit zeros (they are removed) and checks for null rows/columns in WA (they are removed and this includes rows corresponding to zero weights). The user may optionally flag rows that are to be treated as dense or may specify the density of rows that are to be treated as dense. Any such rows are permuted to be the final rows of the cleaned matrix. Finally, it orders the row indices within each column of the permuted matrix in increasing order (this is required by MA85_factor). The user's data is unaltered.

To perform checking, a call of the following form should be made:

```
call MA85_check_matrix(m_in, n_in,  ptr_in, row_in, val_in, m, n, md, ptr, row, val,  &
     rowmap, colmap, control, info[, rowflag, density, weight_in, weight, b_in, b])
```

`m_in` is an `INTENT(IN)` scalar of type `INTEGER` that must hold the number of rows of A.

`n_in` is an `INTENT(IN)` scalar of type `INTEGER` that must hold the number of columns of A.

`ptr_in` is an `INTENT(IN)` rank-1 array of type `INTEGER` and size `n_in+1`. For `j = 1,2,...,n_in`, `ptr_in(j)` must hold the position in `row` of the first entry in column `j` of A and `ptr_in(n_in+1)` must be set to one more than the number of matrix entries.

`row_in` is an `INTENT(IN)` rank-1 array of type `INTEGER` and size at least `ptr_in(n_in+1)-1`. It must hold the row indices of the entries of A with the row indices for the entries in column 1 preceding those for column 2, and so on.

`val_in` is an `INTENT(IN)` rank-1 array of package type and size at least `ptr_in(n_in+1)-1`. `val_in(k)` must hold the value of the entry in `row_in(k)`, `k = 1,2,...,ptr_in(n_in+1)-1`.

`m` is an `INTENT(OUT)` scalar of type `INTEGER` that, on exit, holds the row dimension of the cleaned matrix (null rows of WA removed).

`n` is an `INTENT(OUT)` scalar of type `INTEGER` that on exit, holds the column dimension of the cleaned matrix (null columns of WA removed).

`md` is an `INTENT(OUT)` scalar of type `INTEGER` that, on exit, holds the number of rows that are to be treated as dense. These rows are the last `md` rows of the output matrix. If both the optional arguments `rowflag` and `density` are absent then `md` is set to `0`.

`ptr` is an `INTENT(OUT)` allocatable rank-1 array of type `INTEGER`. On exit, for `j = 1,2,...,n`, `ptr(j)` is the position in `row` of the first entry in column `j` of the cleaned and permuted matrix and `ptr(n+1)` is set to one more than the number of entries in the cleaned matrix.

`row` is an `INTENT(OUT)` allocatable rank-1 array of On exit, the first `ptr(n+1)-1` entries hold the row indices of the cleaned and permuted matrix, with the row indices for the entries in column 1 preceding those for column 2, and so on, and within each column the indices are in increasing order.

`val` is an `INTENT(OUT)` allocatable rank-1 array of package type. On exit, `val(k)` holds the value of the entry in `row(k)`, `k = 1,2,...,ptr(n+1)-1`.

`rowmap` is an `INTENT(OUT)` rank-1 array of type `INTEGER` and size `m_in`. On exit, if `i` is the index of a row of the user-supplied matrix A ($1 \leq$ `i` $\leq$ `m_in`), then `rowmap(i)` holds its index in the cleaned and permuted matrix or is set to `0` if row `i` has been removed.

`colmap` is an `INTENT(OUT)` rank-1 array of type `INTEGER` and size `n_in`. On exit, if `j` is the index of a column of the user-supplied matrix A ($1 \leq$ `j` $\leq$ `n_in`), then `colmap(j)` holds its index in the cleaned and permuted matrix or is set to `0` if column `j` has been removed.

`control` is an `INTENT(IN)` scalar of type `MA85_control` (see Section 2.6).

`info` is an `INTENT(OUT)` scalar of type `MA85_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.7.

`rowflag` is an optional `INTENT(IN)` rank-one array of type `LOGICAL` and size `m_in`. If present, `rowflag(i)` should be set to `.true.` if row `i` of A is to be treated as dense and to `.false.` otherwise. Rows flagged as dense will be permuted to be the last `md` rows.

density is an optional INTENT(IN) scalar of package type that is only accessed if rowflag is not present. In this case, rows of A that have density greater than or equal to density are treated as dense (the density is the number of entries in the row divided by n); such rows are permuted to be the last md rows. If density is less than or equal to zero, no rows are treated as dense; values of density that are greater than 1.0 are treated as 1.0.

weight_in is an optional INTENT(IN) rank-one array of package type and size m_in. If present, weight_in(i) must hold the absolute value of the weight for row i of A, $1 \leq$ i $\leq$ m_in. If weight_in(i) is equal to zero, row i will be removed from the cleaned and permuted matrix.

weight is an optional INTENT(OUT) rank-one array of package type. It must be present if weight_in is present. On exit, weight(i), $1 \leq$ i $\leq$ m holds the weight for row i of the cleaned and permuted matrix.

b_in is an optional INTENT(INOUT) rank-one array of package type and size m_in that, if present, must hold a vector b corresponding to the matrix A input by the user.

b is an optional INTENT(OUT) rank-one array of package type. It must be present if b_in is present. On exit, entries corresponding to null rows of WA are removed and the vector is permuted so that b can be passed unchanged to MA85_solve.

2.5.4 To clean b and/or W

After a call to MA85_check_matrix for which m_in $\neq$ m, additional vectors b and weights W can be cleaned using a call of the following form:

```
call MA85_cleanb(m_in, m, rowmap, control, info, weight_in, weight, b_in, b)
```

m_in, m, rowmap are INTENT(IN) and must be passed as output from the call to MA85_check_matrix.

control is an INTENT(IN) scalar of type MA85_control (see Section 2.6). Only the diagnostic printing controls are used.

info is an INTENT(INOUT) scalar of type MA85_info (see Section 2.7) Only the component info%flag is accessed.

weight_in, weight, b_in, b: see Section 2.5.3.

2.5.5 The factorize phase

To perform the factorization, a call of the following form should be made:

```
call MA85_factor(approach, m, n, md, ptr, row, val, keep, control, info[, weight])
```

approach is an INTENT(IN) scalar of type INTEGER If approach $\leq$ 1, then Approach (i) employing the Cholesky code HSL_MA87 is used; if approach $\geq$ 2, then Approach (ii) employing the symmetric indefinite solver HSL_MA97 is used.

m, n, md, ptr, row, val are INTENT(IN) and must be passed as output from the call to MA85_check_matrix. Note in particular that in the array row, the row indices within each column **must** be in increasing order with the md dense rows (md$\geq$0) last (**no checks** are made by MA85_factor on the matrix data).

keep is an INTENT(OUT) scalar of type MA85_keep. It is used to hold data about the problem being solved and must be passed unchanged to MA85_solve.

control is an INTENT(IN) scalar of type MA85_control. Its components provide parameters used during the factorization; see Section 2.6 for details.

info is an INTENT(INOUT) scalar of type MA85_info. Its components provide information about the execution of the subroutine, as explained in Section 2.7.

weight is an optional INTENT(IN) rank-one array of package type and size m that, if present, must be set so that the absolute value of weight(i) is the weight for row i of the matrix input using ptr, row, val. If weight_in was present on the call to MA85_check_matrix (or MA85_cleanb), weight should be passed as output from that call. Entries of weight may not be zero (no checks are made). If weight is not present, $W = I$ is used.

2.5.6 The solve phase

To compute the solution x, a call of the following form may be made for each vector b:

```
call MA85_solve(m, n, md, ptr, row, b, x, res, keep, control, info)
```

m, n, md, ptr, row are INTENT(IN) and must be passed as output from the call to MA85_check_matrix.

b is an INTENT(IN) rank-one array of package type and size m that must hold the vector b. If b_in was present on the call to MA85_check_matrix (or MA85_cleanb), b should be passed as output from that call.

x is an INTENT(IN) rank-one array of package type and size n that on exit holds the solution vector.

res is an INTENT(OUT) rank-one array of package type and size m. On exit, it holds the weighted residual vector.

keep is an INTENT(IN) scalar of type MA85_keep that must be passed unchanged since the call to MA85_factor.

control is an INTENT(IN) scalar of type MA85_control (see Section 2.6).

info is an INTENT(INOUT) scalar of type MA85_info. Its components provide information about the execution of the subroutine, as explained in Section 2.7.

2.5.7 Final call

Once all calls MA85_solve are complete, a call of the following form should be made to deallocate allocatable components of keep.

```
call MA85_finalise(keep,info)
```

keep is an INTENT(INOUT) scalar of type MA85_keep that must be passed unchanged from the call to MA85_solve. On exit, allocatable components will be deallocated.

info is an INTENT(IN) scalar of type MA85_info. Only the components flag and stat are accessed (see Section 2.7).

2.5.8 All-in-one call

The user may make a single call of the following form that combines the calls to MA85_check_matrix, MA85_factor, MA85_solve, MA85_finalise to solve the least squares problem. Note that the input data is unchanged (the routine makes a local copy of the cleaned matrix).

```
call MA85_LS(approach, m_in, n_in, ptr_in, row_in, val_in, b_in, x, res, &
     control, info[, rowflag, density, weight_in])
```

approach, control, info: see Section 2.5.5.

m_in, n_in, ptr_in, row_in, val_in, b_in, rowflag, density, weight_in: see Section 2.5.3.

x is an INTENT(OUT) rank-one array of package type and size n_in. On exit, it holds the solution vector (if column j of the input matrix is null, then x(j) is set to zero).

res is an INTENT(OUT) rank-one array of package type and size at least m_in. On exit, it holds the weighted residual vector (if row i of WA is null, then res(i) is set to zero).

2.6 The control derived data type

The derived data type MA85_control is used to hold parameters that control the action. The components are automatically given default values in the definition of the type.

Components that control printing

diagnostics_level is a scalar of type INTEGER that is used to control the level of diagnostic printing. The different levels are:

- < 0 No printing.
- = 0 Error and warning messages only.
- = 1 As 0, plus basic diagnostic printing.
- = 2 As 1, plus some additional diagnostic printing.
- = 3 As 2, plus all entries of user-supplied arrays.

The default is diagnostics_level=0.

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

unit_error is a scalar of type INTEGER with default value 6 that is used as the output stream for error messages. If it is negative, these messages will be suppressed.

unit_warning is a scalar of type INTEGER with default value 6 that is used as the output stream for warning messages. If it is negative, these messages will be suppressed.

Other components

alpha is a scalar of package type with default value 0.0. The absolute value of alpha holds the regularization parameter α. For Approach (i), a small nonzero value of alpha may avoid break down of the Cholesky factorization if the normal matrix C (or C_s if md > 0) is rank deficient. In particular, if the sparse part A_s has null columns then the user should set alpha > 0.0. For Approach (ii), a small nonzero value of alpha may improve the performance of HSL_MA97 (in terms of the number of delayed pivots, operation counts, number of entries in the factor as well as the time).

beta is a scalar of package type with default value 1.0. The absolute value of beta holds the scaling parameter β.

control87 is a scalar of type MA87_control. It is used if approach $= 1$. Default settings are used, except printing is turned off. If printing from HSL_MA87 is required, the user should set control%diagnostics_level$\geq$1. Full details of the HSL_MA87 control parameters are given in the specification documentation for HSL_MA87.

control97 is a scalar of type MA97_control. It is used if approach = 2. Default settings are used, with the exception of turning off printing and the ordering (which is controlled by iorder (see below)). If printing from HSL_MA97 is required, the user should set control%diagnostics_level≥1. Full details of the HSL_MA97 control parameters are given in the specification documentation for HSL_MA97.

curvature is a scalar of package type. It is used to control the smallest allowable curvature within CGLS, thus is only used if approach = 1 and control%alpha > 0.0. It has default value {1.0, and in this case the smallest allowable curvature is *nu*, where *u* is the relative machine precision. Values of control%curvature that are less than or equal to zero are treated as if they were the default {1.0.

delta1 and delta2 are scalars of package type that control the stopping criteria when iterative refinement or CGLS is used (approach = 1 only). They have default value $\sqrt{(u)}$, where u is the relative machine precision. The following stopping rules are used:

- C1: Stop if $\|r_k\|_2 <$ delta1.
- C2: Stop if
$$\frac{\|A^T W r_k\|_2}{\|r_k\|_2} < \frac{\|A^T W r_0\|_2}{\|r_0\|_2} * \texttt{delta2},$$

where $r_k = W(b - Ax_k)$ is the computed residual on the k-th iteration. Refinement terminates if either C1 or C2 is satisfied or the maximum number of iterations (control%maxit_IR or control%maxit_CGLS) is exceeded.

delta_gmres is a scalar of package type that controls the GMRES stopping criteria (approach = 2 only). It has default value $\sqrt{(u)}$, where u is the relative machine precision. GMRES terminates if either the maximum number of iterations is exceeded (control%maxit_GMRES) or

$$\frac{\|Ky_k - c\|_2}{\|c\|_2} < \texttt{delta\_gmres},$$

where y_k is the solution of (1.6) on the k-th iteration.

iorder is a scalar of type INTEGER with default value 1 that controls preordering of the matrix prior to the application of the sparse direct solver HSL_MA87 or HSL_MA97. Options available are:

1 An approximate minimum degree (AMD) ordering (computed using HSL_MC68).

2 METIS (nested dissection) ordering with default settings. If METIS is not supplied and this option is requested, no ordering is performed.

For all other values, no ordering is performed.

limit_colC is a scalar of type INTEGER with default value -1 that is only used if approach = 1. If a column of C (or C_s if md > 0) has more than limit_colC entries, the computation terminates with an error flag set. If limit_colC is negative, the limit is n.

limit_C is a scalar of type INTEGER with default value -1 that is only used if approach = 1. If C (or C_s if md > 0) has more than limit_C entries (upper and lower triangular parts), the computation terminates with an error flag set. If limit_C is negative, the limit is huge(1).

maxit_IR is a scalar of type INTEGER with default value 10. It is used if approach = 1 and control%alpha = 0.0 to hold the maximum number of steps of iterative refinement.

maxit_CGLS is a scalar of type INTEGER with default value 0. It is used if approach = 1 and control%alpha > 0.0 to hold the maximum number of steps of preconditioned CGLS (in this case, if control%maxit_CGLS > 0, the solution of the unregularized problem (1.1) is computed).

maxit_GMRES is a scalar of type INTEGER with default value 0. It is used if approach = 2 and control%alpha > 0.0 to hold the maximum number of steps of preconditioned GMRES (in this case, if control%maxit_GMRES>0, the solution of the unregularized problem (1.1) is computed).

scale is a scalar of type LOGICAL with default value .true. that controls whether scaling is used. If set to .true., we solve

$$\min_z \|W(ASz-b)\|_2, \qquad x = Sz,$$

where S is the diagonal matrix with entries S_{ii} satisfying $S_{ii}^2 = 1/\|WAe_i\|_2$ (e_i is the $i-$th unit vector).

weight_tol is a scalar of package type with default value 0.0. It is used only by the checking routines. Entries of the array weight that have absolute value less than or equal weight_tol are treated as zero (and leads to rows of WA being treated as zero).

2.7 The derived data type for holding information

The components of the derived data type MA85_info are used to hold information.

Information returned for approach = 1

info87 is a scalar of type MA87_info. Its components hold information returned by HSL_MA87; full details are given in the specification documentation for HSL_MA87. In particular, info87%flag holds the exit status for HSL_MA97.

nnz_C is a scalar of type INTEGER that, on exit from MA85_factor (and MA85_LS), holds the number of entries in the normal matrix.

num_factor is a scalar of type INTEGER(long) that, on exit from MA85_factor (and MA85_LS), holds the number of entries that in the Cholesky factor of the normal matrix.

num_flops is a scalar of type INTEGER(long) that, on exit from MA85_factor (and MA85_LS), holds the number of floating-point operations used to perform the factorization of the normal matrix.

Information returned for approach = 2

info97 is a scalar of type MA97_info. Its components hold information returned by HSL_MA97; full details are given in the specification documentation for HSL_MA97. In particular, info97%flag holds the exit status for HSL_MA97.

num_delay is scalar of type INTEGER that, on exit from MA85_factor (and MA85_LS), holds the number of eliminations that were delayed, that is, the total number of fully-summed variables because of stability considerations. Using a regularization parameter $\alpha > 0$ or using a smaller pivot tolerance u may reduce the number of delayed eliminations and speed up the factorization and improve the sparsity of the factors.

num_factor is scalar of type INTEGER(long) that, on exit from MA85_factor (and MA85_LS), holds the number of entries in the L factor of the augmented system.

num_flops is scalar of type INTEGER(long) that, on exit from MA85_factor (and MA85_LS), holds the number of floating-point operations used to compute the factorization.

Other information

flag is a scalar of type INTEGER that gives the exit status of the algorithm (details in Section 2.8).

flag68 is a scalar of type INTEGER that, on exit from MA85_factor (and MA85_LS), holds the exit status on return from the ordering routine HSL_MC68 (and is set to 0 if HSL_MC68 is not used).

flag_potrf is a scalar of type INTEGER that, on exit from MA85_factor (and MA85_LS), holds the exit status on return from the LAPACK routine _potrf

iter_refine is a scalar of type INTEGER that, on exit from MA85_solve (and MA85_LS), holds the number of steps of iterative refinement that have been performed (normal = .true. with control%alpha = 0.0).

iter_cgls is a scalar of type INTEGER that, on exit from MA85_solve (and MA85_LS), holds the number of iterations of CGLS that have been performed (normal = .true. with control%alpha $\neq$ 0.0).

iter_gmres is a scalar of type INTEGER that, on exit from MA85_solve (and MA85_LS), holds the number of iterations of GMRES that have been performed (normal = .false.).

m is a scalar of type INTEGER that, on exit from MA85_LS, holds the row dimension of the cleaned least squares matrix.

md is a scalar of type INTEGER that, on exit from MA85_LS, holds the number of rows in the cleaned least squares matrix that are treated as dense.

n is a scalar of type INTEGER that, on exit from MA85_LS, holds the column dimension of the cleaned least squares matrix.

ndup is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of duplicated indices found in row.

nnz_C is a scalar of type INTEGER that, on exit from MA85_factor (and MA85_LS) with approach = 1, holds the number of entries in the lower triangular part of the normal matrix that is factorized using HSL_MA87 (that is, C or C_s if $\mathtt{m_d} > 1$).

noor is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of out-of-range indices found in row.

norm_ATW2b is a scalar of package type that, on exit from MA85_solve (and MA85_LS), holds $\|A^TW^2b)\|_2$.

norm_ATr is a scalar of package type that, on exit from MA85_solve (and MA85_LS), holds $\|A^TW^2(Ax-b)\|_2$.

norm_r is a scalar of package type that, on exit from MA85_solve (and MA85_LS), holds $\|W(Ax-b)\|_2$.

norm_Wb is a scalar of package type that, on exit from MA85_solve (and MA85_LS), holds $\|Wb\|_2$.

nzero is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of explicit zero entries found and removed from A.

nzero_col is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of null columns in WA.

nzero_row is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of null rows in WA.

nzero_weight is a scalar of type INTEGER that, on exit from MA85_check_matrix (and MA85_LS), holds the number of zero weights found in weight. Zero weights result in an error flag being set and the computation terminates.

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

test1 is a scalar of type LOGICAL that, on exit from MA85_check_matrix (and MA85_LS), is set to .true. if the stopping condition C1 (see control%delta1) is satisfied.

test1 is a scalar of type LOGICAL that, on exit from MA85_check_matrix (and MA85_LS), is set to .true. if the stopping condition C2 (control%delta2) is satisfied.

2.8 Warning and error messages

A successful return from a subroutine in the package is indicated by info%flag having the value zero. A negative value is associated with an error message that by default will be output on unit control%unit_error.

Possible negative values are:

-1 memory allocation failed. The stat parameter is returned in info%stat.

-2 The array row_in is too small.

-3 The array val_in is too small.

-4 Either m_in $<$ 1 or n_in $<$ 1 or, after the removal of null rows and columns of *WA*, either m $<$ 1 or n $<$ 1.

-5 Error in the array ptr.

-6 All columns of *WA* are null columns (this could happen if all the row indices are out-of-range or if all entries of *A* are zero or because of zero weights).

-7 Out-of-range indices in row. The number of such entries is given in info%noor.

-8 Duplicated entries in row. The number of such entries is given in info%ndup.

-9 $\mathtt{m_d} \geq \mathtt{m}$ or $\mathtt{m} - \mathtt{m_d} < \mathtt{n}$.

-10 One or more columns of C (or C_s if $\mathtt{m_d} > 1$) has more than control%limit_colC entries.

-11 C (or C_s if $\mathtt{m_d} > 1$) has more than control%limit_C entries.

-12 Error returned by HSL_MA87 that the normal matrix it was applied to is not positive definite. Using a shift control%alpha>0.0 may avoid this error.

-13 Error returned by the sparse solver that IEEE infinities found during the factorization.

-14 Unexpected error returned by _potrf. Using a shift control%alpha>0.0 may avoid this error.

-15 Memory deallocation failed (MA85_finalise and MA85_LS only). The stat parameter is returned in info%stat.

-16 METIS ordering requested but not available.

-17 Unexpected error from HSL_MA97. Further information provided by info%info97%flag.

-18 One or more expected optional arguments (weight or b) is missing.

-19 One or more null columns in A_s and control%alpha=0.0. (approach = 1). The user may set control%alpha>0.0 and recall MA85_factor (or MA85_LS).

Positive values for info%flag are associated with a warning. A lower value flag may be overwritten by a higher value flag. Possible positive values are:

+1 Explicit zeros have been removed from val.

+2 Null rows have been found in *WA* and removed.

+3 Null columns have been found in *WA* and removed.

+4 A problem was encountered when computing the requested ordering. The original ordering is retained and the error flag returned by HSL_MC68 is given by info%flag68.

+5 Convergence of iterative refinement not achieved within control%maxit_IR iterations.

+6 Convergence of CGLS not achieved within control%maxit_CGLS iterations.

+7 Convergence of GMRES not achieved within control%maxit_GMRES iterations.

+8 Warning returned by HSL_MA97. Further information provided by info%info97%flag.

+9 Breakdown of CGLS (curvature encountered too small to continue) so requested accuracy not achieved.

3 GENERAL INFORMATION

Input/output: Error messages on stream control%unit_error and warning and messages on stream control%unit_warning; printing is suppressed if the relevant stream number is negative.

Restrictions: m≥n≥1, $\mathtt{m_d} < \mathtt{m}$ and $\mathtt{m} - \mathtt{m_d} \geq \mathtt{n}$.

4 METHOD

MA85_check_matrix checks the user-supplied matrix data. Out-of-range entries and/or duplicates leads to an error flag being set and the computation terminates. Entries with value zero are removed, along with null rows or columns of WA. Dense rows are optionally identified and permuted to be the last m_d rows. The entries within each column of the permuted and cleaned matrix are ordered by increasing row index. If a vector b is input, entries corresponding to the removed rows of WA are also removed. Checking of the data is recommended as no further checking of the user-supplied data is carried out and errors (including explicit zeros and/or null rows/columns) may lead to unpredictable behaviour later in the computation. If the user wishes to clean further b and/or W, calls to MA85_cleanb may be made after the call to MA85_check_matrix.

Approach (i) If $m_d = 0$ (no dense rows) the (regularized) normal matrix $C+\alpha I$, $C = A^T W^2 A$, $\alpha =$control%alpha, is formed as a sparse matrix and factorized using HSL_MA87. If control%alpha=0.0 (or is very small), the factorization may breakdown (for example, if A is close to being rank deficient). In this case, the user can increase control%alpha and restart the factorization. During the call to MA85_solve, if control%alpha=0.0, iterative refinement may be used to improve the the solution. If control%alpha> 0.0, then the computed factors may optionally be used as a preconditioner for CGLS to recover the solution of the unregularized problem (1.1).

If $m_d > 1$, the solution is obtained from the $(n+m_d)\times(n+m_d)$ augmented system

$$\begin{pmatrix} C_s+\alpha I & A_d^T W_d \\ W_d A_d & -I \end{pmatrix} \begin{pmatrix} x \\ W_d A_d x \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix},$$

where $C_s = A_s^T W_s^2 A_s$, $\alpha =$control%alpha, and $c = A_s^T W_s^2 b_s + A_d^T W_d^2 b_d$. The Cholesky factorization $C_s+\alpha I = L_s L_s^T$ is computed using HSL_MA87 (a fill-reducing ordering chosen using the parameter control%iorder may be applied before the factorization is computed). This yields a signed Cholesky factorization

$$\begin{pmatrix} P_s & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} C_s+\alpha I & A_d^T W_d \\ W_d A_d & -I \end{pmatrix} \begin{pmatrix} P_s^T & 0 \\ 0 & I \end{pmatrix} = \begin{pmatrix} L_s & \\ B_d & L_d \end{pmatrix} \begin{pmatrix} I & \\ & -I \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ & L_d^T \end{pmatrix}, \tag{4.1}$$

where P_s is a permutation matrix,

$$L_s B_d^T = P_s A_d^T W_d \tag{4.2}$$

and

$$L_d L_d^T = I + B_d B_d^T. \tag{4.3}$$

Equation (4.2) is solved using MA87_solve and, if $m_d > 1$, the factorization (4.3) is computed using the Lapack routine _potrf.

Approach (ii) If $m_d = 0$ (respectively, $m_d > 1$, the lower triangular part of the augmented system matrix K (1.6) (respectively, K_r (1.8)) is formed as a sparse matrix and factorized using the indefinite solver HSL_MA97. This incorporates ordering for sparsity and for numerical stability. If info%num_delay is large, the factorization time and memory required may be reduced by using a non zero regularization parameter control%alpha. Once the factorization has been computed, GMRES may be employed to obtain the solution of the unregularized problem (1.1).

5 EXAMPLE OF USE

Suppose we wish to compute the vector x that minimizes $\|W(Ax-b)\|_2$, where

$$A = \begin{pmatrix} 1. & 0. & 4. \\ 0. & 1. & 0. \\ 0. & 0. & 5. \\ 2. & 3. & 1. \end{pmatrix} \quad W = \begin{pmatrix} 2. & & & \\ & 1. & & \\ & & 2. & \\ & & & 1. \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 1. \\ 1. \\ 1. \\ 1. \end{pmatrix}.$$

We may use the following code:

```
program ma85_spec_double

  ! Simple example to illustrate use of HSL_MA85

  use hsl_ma85_double
  implicit none

  integer, parameter :: wp = kind(1.0d0)
  real(wp), parameter :: zero = 0.0_wp

  type(ma85_control) :: control
  type(ma85_info) :: info
  type(ma85_keep) :: keep

  integer,  allocatable :: ptr(:),row(:)
  integer,  allocatable :: ptr_in(:),row_in(:)
  integer,  allocatable :: rowmap(:),colmap(:)
  logical,  allocatable :: rowflag(:)
  real(wp), allocatable :: b(:),val(:),weight(:)
  real(wp), allocatable :: b_in(:),val_in(:),weight_in(:)
  real(wp), allocatable :: res(:),x(:)

  integer :: approach,m,m_in,md,n,n_in,nz

  ! Read in the matrix data
  read (*,*) m_in,n_in,nz

  allocate (ptr_in(n_in+1),row_in(nz),val_in(nz),weight_in(m_in),b_in(m_in), &
            res(m_in),x(n_in),rowmap(m_in),colmap(n_in),rowflag(m_in))
```

```
   read (*,*) ptr_in(1:n_in+1)
   read (*,*) row_in(1:nz)
   read (*,*) val_in(1:nz)
   read (*,*) weight_in(1:m_in)
   read (*,*) b_in(1:m_in)

   ! set rowflag  (dense row is last)
   rowflag(1:3) = .false.
   rowflag(4) = .true.

   call ma85_check_matrix(m_in, n_in, ptr_in, row_in, val_in,   &
        m, n, md, ptr, row, val, rowmap, colmap, control, info,     &
        weight_in=weight_in, weight=weight, b_in=b_in, b=b)

   if(info%flag.lt.0) then
      print *, "ma85_check_matrix failed with error ", info%flag
      go to 99
   end if
   deallocate (ptr_in,row_in,val_in,weight_in,b_in)

   approach = 1
   call ma85_factor(approach, m, n, md, ptr, row, val, keep, control, info, &
        weight=weight)

   if (info%flag.lt.0) then
      print *, "ma85_factor failed with error ", info%flag
      go to 99
   end if

   call MA85_solve(m, n, md, ptr, row, b, x, res, keep, control, info)

   if (info%flag.lt.0) then
      print *, "ma85_solve failed with error ", info%flag
      go to 99
   end if

   write (*,'(a,3es12.3)') ' Solution = ', x(1:n)
   write (*,'(a,es12.3)') ' Residual ||r|| = ',info%norm_r

99 call ma85_finalise(keep,info)

   deallocate (ptr,row,val,weight,b,res,x,rowmap,colmap)

 end program ma85_spec_double
```

With the input data:

```
  4  3  7
  1  3  5  8
  1  4  2  4  1  3  4
```

```
  1.0  2.0  1.0  3.0  4.0  5.0  1.0
  2.0  1.0  2.0  1.0
  5.0  2.0  5.0  9.0
```

we obtain the following output:

```
 Solution =    2.350E-02   3.211E-01   2.158E-01
 Residual ||r|| =    7.670E-01
```