Science and Technology Facilities Council

HSL_MA79

PACKAGE SPECIFICATION HSL

1 SUMMARY

HSL_MA79 is a **mixed precision** sparse symmetric solver for solving one or more linear systems $\mathbf{AX} = \mathbf{B}$. A factorization of $\mathbf{A}$ using single precision (that is, 32-bit real arithmetic) is performed using a direct solver (MA57 or HSL_MA77) and then refinement (iterative refinement and, in some cases, FGMRES) in double precision (that is, 64-bit real arithmetic) is used to recover higher accuracy. This technique is termed a mixed precision approach. If refinement fails to achieve the requested accuracy, a double precision factorization is performed.

Use of single precision arithmetic substantially reduces the amount of data that is moved around within a sparse direct solver, and on a number of modern architectures, it is currently significantly faster than double precision computation. Thus HSL_MA79 offers the potential of obtaining a solution to $\mathbf{AX} = \mathbf{B}$ to double-precision accuracy more rapidly than using a direct solver from HSL in double precision. HSL_MA79 is primarily designed for solving very large systems.

ATTRIBUTES — Version: 1.2.0 (4 April 2013) **Types:** Real (double). **Calls:** HSL_MA54, MA57, HSL_MA64, HSL_MA77, MC30, MC34, MC64, MC77, MI15, HSL_OF01, HSL_ZD11. **Date:** June 2008. **Origin:** J. D. Hogg and J. A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Parallelism:** May use OpenMP through HSL_MA77. **Remark:** The development of HSL_MA79 was supported by EPSRC grants EP/F006535/1 and EP/E053351/1.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement

```
USE HSL_MA79_double
```

The following procedures are available to the user:

- MA79_factor_solve accepts the matrix $\mathbf{A}$, the right-hand sides $\mathbf{B}$, and the required accuracy. Based on this data, it decides which sparse direct solver to use. The matrix $\mathbf{A}$ is factorized, the linear system solved and, if necessary, refinement used to achieve the requested accuracy. The matrix factorization is retained for further solves.

- MA79_refactor_solve allows the user to use the information gained from a previous call to MA79_factor_solve to (potentially) reduce the time required to factorize and solve $\mathbf{AX} = \mathbf{B}$, for a new (but similar) matrix $\mathbf{A}$. The sparsity pattern of $\mathbf{A}$ must be unchanged since the call to MA79_factor_solve; only the numerical values of the entries of $\mathbf{A}$ and $\mathbf{B}$ may have changed. The matrix factorization overwrites the existing factorization.

- MA79_solve uses the computed factors generated by MA79_factor_solve (or MA79_refactor_solve) to solve further systems $\mathbf{AX} = \mathbf{B}$. Multiple calls to MA79_solve may follow a call to MA79_factor_solve (or MA79_refactor_solve) but these will always use the precision of the factorization from the previous call to MA79_factor_solve (or MA79_refactor_solve) — if a solve fails to reach the desired accuracy with single precision factors the user should call MA79_refactor_solve with control%prec = 2 to force a double precision factorization. Multiple calls to MA79_solve in mixed precision mode may result in a slower overall solution time than the use of double precision throughout.

- MA79_finalise should be called after all other calls are complete for a problem. It deallocates the components of the derived data types and discards the matrix factors.

2.2 OpenMP

OpenMP is used by the HSL_MA77 package to provide parallelism for shared memory environments. Parallel speedups will only be occur if the solver HSL_MA77 is chosen — this can be forced by setting control%solver = 2 (see Section 2.6.8). The number of threads may be controlled at runtime by setting the environment variable OMP_NUM_THREADS.

If OpenMP is available then it should be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp).

2.3 MeTiS

The MA57 package uses the MeTiS graph partitioning library available from the University of Minnesota website. If MeTiS is not available then the user must compile with the supplied replacement subroutine METIS_NodeND.

2.4 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types MA79_control, MA79_info, and MA79_keep. The following pseudocode illustrates this.

```
use HSL_MA79_double
...
type (MA79_control) :: control
type (MA79_info) :: info
type (MA79_keep) :: keep
...
```

The components of MA79_control and MA79_info are explained in Sections 2.6.8 and 2.6.9, respectively. The components of MA79_keep are private and are used to pass data between the subroutines of the package.

2.5 Performance notes

2.5.1 Floating point underflows

The user is advised that performance in single precision can be adversely affected on x86 CPUs if a large number of floating-point underflows occur. To avoid a cascade of floating-point underflows, it may be helpful to enable floating-point flush to zero (FTZ) behaviour by either using an appropriate flag on the Fortran compiler or a system call; common compiler options are detailed below.

As these compiler flags set the mode for the program globally, users should be careful to ensure that the numerical stability of other code is not affected.

g95	Set the environment variable G95_FPU_NO_DENORMALS to true before running the compiled executable.
NAG f95	-ieee=nonstd
ifort	On by default (x86, x86_64); can be disabled with -no-ftz
gfortran	None; use C functions.

Fine-tuned control can be effected through the use of the following C functions on processors supporting SSE instructions.

```
/* Sets Flush to Zero Mode */
void set_x86_ftz () {
   _MM_SET_FLUSH_ZERO_MODE (_MM_FLUSH_ZERO_ON);
}
```

```
/* Restores IEEE compliant Denormals */
void unset_x86_ftz () {
   _MM_SET_FLUSH_ZERO_MODE (_MM_FLUSH_ZERO_OFF);
}
```

2.6 Argument lists and calling sequences

2.6.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments, which always follow the argument info. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that all optional arguments be called by keyword, not by position**.

2.6.2 Integer and real kinds

INTEGER(short) denotes default INTEGER.

INTEGER(long) denotes INTEGER(kind=selected_int_kind(18)).

REAL(sp) denotes single precision real and REAL(dp) denotes double precision real.

2.6.3 32-bit and 64-bit architectures

By default, it is assumed that the architecture is 32-bit. The parameter control%bits should be set to 64 if the user is running on a 64-bit architecture. On a 32-bit architecture, the maximum size of a rank-1 REAL(dp) array that can be allocated is taken to be huge(0_short)/8, where huge is the Fortran enquiry function. On a 64-bit architecture, it is taken to be huge(0_long)/8.

2.6.4 The factorization subroutine

```
call MA79_factor_solve(A, psdef, nrhs, lrhs, rhs, accuracy, keep, control, &
                       info[, order, resid, sres_sub])
```

A is a scalar INTENT(INOUT) argument of type ZD11_TYPE. On entry, the INTEGER(short) scalar component A%n must be set to the order of **A**. In addition, the following components must be allocated and set:

- ptr is an INTEGER(short) rank-one array that must be allocated to be of size at least A%n+1. The first A%n entries must be set to hold the starting position of each column of the lower triangular part (including the diagonal) of **A** in a compressed sparse column storage scheme. A%ptr(A%n+1) must hold the index after the last entry of the matrix.
- row is an an INTEGER(short) rank-one array that must be allocated to be of size at least the number of entries in **A** (upper and lower triangular parts). The first A%ptr(n+1)-1 entries must be set to hold the column indices of the entries of the lower triangular part of **A** (including the diagonal) in a compressed sparse column storage scheme (within each column, the entries may be any order).
- val is a REAL(dp) rank-one array that must be allocated to be of size at least the number of entries in **A** (upper and lower triangular parts). The first A%ptr(n+1)-1 entries must be set to hold the values of the entries of the lower triangular part of **A** (including the diagonal) in a compressed sparse column storage scheme in the same order as in row.

Any entries in the strictly upper triangular part of **A** will be removed and a warning issued; duplicate entries will be summed. On exit, the above components of A will contain the matrix in expanded form (upper and lower triangular parts). A%ne will contain the number of entries in **A** and A%ptr, A%row and A%val will hold **A** in compressed sparse column format, with the entries in each column ordered by increasing row index.
Restriction: A%n≥1

psdef is a scalar INTENT(IN) argument of type LOGICAL. It should be set to .true. if **A** is known to be positive definite and to .false. otherwise. If pos_def = .true., the factorization will generally be faster (and the matrix factor may be sparser) because no numerical pivoting for stability is performed. However, if a matrix is falsely claimed to be positive definite, the factorization may have to restart the factorization, resulting in a greater total solution time than if psdef was initially set to .false..

nrhs is a scalar INTENT(IN) argument of type INTEGER(short) that holds the number of right-hand sides. If nrhs= 0 then only a factorization is performed. **Restriction:** nrhs $\geq$ 0.

lrhs is a scalar INTENT(IN) argument of type INTEGER(short) that holds the first extent of rhs. **Restriction:** lrhs $\geq$ A%n.

rhs is a rank-2 array with extents lrhs and nrhs of INTENT(INOUT) and type REAL(dp). It must be set so that x(1:A%n,i) holds the ith right-hand side. On exit, x(1:A%n,i) will have been overwritten with the solution for the ith right hand side.

accuracy is a scalar INTENT(IN) argument of type REAL(dp) and indicates the desired accuracy of the computed solution. The computed solution is accepted when for each vector i = 1,..., nrhs the scaled residual

$$\beta_i = \frac{\|\mathbf{A}\mathbf{x_i} - \mathbf{b_i}\|_\infty}{\|\mathbf{A}\|_\infty \|\mathbf{x_i}\|_\infty + \|\mathbf{b_i}\|_\infty} \tag{2.1}$$

is less than accuracy. If the optional argument sres_sub is present, it is used to calculate the value β_i rather than equation (2.1). Values of accuracy less than zero will be treated as zero.
Warning: Setting this argument too small will significantly increase the computation time; a value greater than (or equal to) 5×10^{-15} is recommended.

keep is a scalar INTENT(INOUT) argument of type MA79_keep that must be passed unchanged by the user to other routines in the package.

control is a scalar INTENT(IN) argument of type MA79_control (see Section 2.6.8).

info is a scalar INTENT(OUT) argument of type MA79_info. Its components provide information about the execution of the subroutine, as explained in Section 2.6.9.

order is an optional rank-1 array INTENT(IN) argument of type INTEGER(short) and size at least A%n. If present, it must specify the elimination order. Specifically, abs(order(i)) must hold the position of variable i in the pivot sequence. If a 1×1 pivot i is required, the user must set order(i)>0. If a 2×2 pivot involving variables i and j is required, the user must set order(i)<0, order(j)<0 and |order(j)| = |order(i)|+1. Note that 2×2 pivots are only appropriate if the matrix **A** is **not** positive definite. If order is not present, an elimination order will be computed using the analyse phase of MA57. **Note:** If order is present, automatic selection of solvers will not work as well, as statistics from MA57 analysis phase will not be used.

resid is an optional rank-1 array INTENT(OUT) argument of type REAL(dp) and size at least nrhs. On exit, resid(i) contains the value of β_i calculated by (2.1), or if sres_sub is present, the sres argument of sres_sub.

sres_sub is an optional subroutine argument that allows the user to specify the measure of the achieved accuracy for a given solution. If present, it must compute a function $\beta = f(\mathbf{A},\mathbf{X},\mathbf{B},\mathbf{R})$ that is compared to accuracy when testing for termination (see Section 4). For example, it may be used to test convergence in the 2-norm or using a component-wise approach (the latter approach is demonstrated by way of example in Section 5.2).

sres_sub must be an external procedure with the following specification:

```
subroutine sres_sub(A, x, b, r, retain, sres)
   use HSL_ZD11_double
   type(ZD11_type), intent(in) :: A
   real(wp), dimension(:, :), intent(in) :: x
   real(wp), dimension(:, :), intent(in) :: b
   real(wp), dimension(:, :), intent(in) :: r
   real(wp), dimension(:), allocatable, intent(inout) :: retain
   real(wp), dimension(:), intent(out) :: sres
end subroutine
```

A holds a copy of the matrix $\mathbf{A}$ with the same components allocated as for the call to MA79_factor_solve, but expanded to full form (ie both lower and upper entries are present).

x holds the current solutions $\mathbf{X}$.

b holds the right hand sides $\mathbf{B}$.

r holds the residual vectors $\mathbf{R} = \mathbf{B} - \mathbf{AX}$.

retain provides storage space for retaining values between calls to the routine (for example, the norm of $\mathbf{A}$). It is unallocated upon the first call to sres_sub or if the matrix data has changed since the last call. If it is allocated then it will be deallocated by MA79_finalize.

sres must hold on return the vector of β values that will be compared against accuracy when testing for termination.

2.6.5 The refactorization subroutine

After a call to MA79_factor_solve (or MA79_solve), if the user wishes to factorize and solve linear systems for which the matrix $\mathbf{A}$ has the same sparsity pattern as on the last call to MA79_factor_solve but different numerical values, a call of the following form may be made.

```
call MA79_refactor_solve(A, psdef, nrhs, lrhs, rhs, accuracy, keep, &
                         control, info[, resid, sres_sub])
```

A is a scalar INTENT(IN) argument of type ZD11_TYPE. The **only** component that may have been changed by the user since the last call to MA79_factor_solve is A%val. The first A%ne entries must be set to hold the values of the lower and upper triangular entries of $\mathbf{A}$ in a compressed sparse column storage scheme in the same order as in A%row (that is, the entries must be in column order, with the entries in each column in order of increasing row index).

keep is a scalar INTENT(INOUT) argument of type MA79_keep that must be unchanged since the call to HSL_MA79.

For all other arguments, see Section 2.6.4.

2.6.6 The solve subroutine

After the call to MA79_factor_solve (or MA79_refactor_solve), one or more calls of the following form may be made to solve for further right-hand sides.

```
call MA79_solve(A, nrhs, lrhs, rhs, accuracy, keep, control, info[, resid, sres_sub])
```

A is a scalar INTENT(IN) argument of type ZD11_TYPE that must be unchanged since the call to MA79_factor_solve (or MA79_refactor_solve).

keep is a scalar INTENT(INOUT) argument of type MA79_keep that must be unchanged since the call to MA79_factor_solve (or MA79_refactor_solve).

For all other arguments, see Section 2.6.4.

2.6.7 The finalisation subroutine

Once all other calls are complete for a problem or after an error return, a call of the following form should be made to deallocate allocatable components of the structure keep and to close and delete any files opened by the package for the problem:

```
call MA79_finalize(keep, control, info)
```

2.6.8 The derived data type for holding control parameters

The derived data type MA79_control is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are as follows:

Printing and timing controls

print_level is a scalar of type INTEGER(short) that controls the amount of printing. The different levels are:

- $\leq$0 No printing.
- =1 Error messages only.
- =2 As 1, plus warning messages.
- =3 As 2, plus simple diagnostic information.
- =4 As 3, plus basic diagnostic and error information from the HSL packages called by HSL_MA79.
- $\geq$5 As 4, plus extra verbose diagnostic information including from the HSL packages called by HSL_MA79.

The default printing level is print_level = 2.

time_parts is a scalar of type LOGICAL. If time_parts=.true. internal timing routines are used to return runtimes for various parts of the code. See info%timings for details. The default is time_parts = .false..

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

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

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

Controls used by MA79_factor_solve

bits is a scalar of type INTEGER(short) that indicates the machine architecture being used. It should be set to 32 on a 32-bit architecture and to 64 on a 64-bit architecture. The default is bits=32. The value of this parameter will affect the choice between MA57 and HSL_MA77 and whether the operation will complete successfully. In particular, if the maximum frontsize is found to exceed approximately 2^{14}, the direct solver used will be HSL_MA77 and the computation will only complete successfully on a 64-bit machine.
Restriction: bits = 32 or 64.

maxmem is a scalar of type INTEGER(long) and controls the maximum amount of memory available for use by the code. The default is maxmem=0. In this case, when running on 32-bit architecture (control%bits = 32), the maximum amount of memory available is taken to be huge(0_short)/8 and on a 64-bit architecture it is taken to be huge(0_long)/8, where huge is the Fortran enquiry function. **Restriction:** if control%bits = 32, maxmem $\leq$ huge(0_short)/8.

ma57_nemin is a scalar of type INTEGER(short) that controls node amalgamation within MA57. The default is ma57_nemin = 16. The default is used if ma57_nemin < 1.

ma77_nemin is a scalar of type INTEGER(short) that controls node amalgamation within HSL_MA77. The default is ma77_nemin = 8. The default is used if ma77_nemin < 1.

ma77_filename is an array of size 4, type CHARACTER and character length at most 400 that identifies the direct access files used by HSL_MA77. The default is ma77_files = (/ "ma77_imain", "ma77_rmain", "ma77_rwork", "ma77_rwdelay" /). All the names must be different. HSL_MA77 uses primary and secondary files. The name of the primary file is ma77_path(i)//ma77_filename(j) for an element i of ma77_path. Secondary files have names that are constructed by appending 1, 2, ... to this form, perhaps with a different element of ma77_path. Further details are given in the user documentation for HSL_MA77.

ma77_file_size is a scalar of type INTEGER(short) that specifies how long the files used by HSL_MA77 should be in Fortran allocation units. If an I/O error occurs that indicates too many open files, the user should try increasing ma77_file_size. The default is ma77_file_size = 2^{21}.

ma77_path is an allocatable array of type CHARACTER(LEN=400). By default it is unallocated, and in this case it will be treated as the array (""). The user may allocate ma77_path, supplying path names for the direct access files used by HSL_MA77. By supplying more than one path name, some of the files used by HSL_MA77 may reside on different devices.
Note: It is the responsibility of the user to ensure this component is deallocated.

solver is a scalar of type INTEGER(short) that controls the initial choice of direct solver.

0 The code will automatically select which solver to use

1 MA57

2 HSL_MA77

The default is solver = 0. **Restriction:** solver = 0, 1 or 2.

Controls used by MA79_factor_solve **and** MA79_refactor_solve

action_indef is a scalar of type LOGICAL. When the argument psdef to MA79_factor_solve or MA79_refactor_solve is set to .true., but the factorization encounters a negative pivot it takes an action depending on the value of action_indef. If action_indef=.true., the current factorization is terminated and a refactorization of the matrix as indefinite is performed; a warning (info%flag=+6) is issued to indicate this. If action_indef=.false., the factorization is terminated and an error (info%flag=-34) is returned to the user. The default is action_indef=.true..

action_singular is a scalar of type LOGICAL. When MA79 detects that a matrix is singular it takes an action depending on the value of action_singular. If action_singular=.true., the solution process proceeds as normal but a warning (info%flag=+7) is issued to indicate a possible problem. If action_singular=.false., the attempted solution is terminated and an error (info%flag=−35) is returned to the user. The default is action_singular=.true..

ma57_blocking is a scalar of type INTEGER(short) that controls the size of the blocks used by MA57 for calling the BLAS. The default value is ma57_blocking = 16. **Restriction:** ma57_blocking > 0.

ma77_nb is a scalar of type INTEGER(short) that controls the block size used within the kernel factorization codes called by HSL_MA77. The default is ma77_nb = 150. **Restriction:** ma77_nb > 0.

fallback_double is a scalar of type LOGICAL. If a mixed precision factorization fails to achieve the desired accuracy in MA79_factor_solve or MA79_refactor_solve (but not in MA79_solve) then the behaviour is determined by fallback_double. If fallback_double=.true., refactorization in double precision is automatically performed. Otherwise a warning is issued (info%flag=+8) and the solution with smallest β is returned. The default value is fallback_double=.true..

prec is a scalar of type INTEGER(short) that controls the precision to be used by the factorization phase of the direct solver.

1 Single precision arithmetic (this should be selected for mixed precision computation).

2 Double precision arithmetic (in this case, the whole computation is carried out in double precision).

The default is prec = 1. **Restriction:** prec = 1 or 2.

scaling is a scalar of type INTEGER(short) that controls scaling of the matrix **A**. It can take the following values:

$\leq$0 No scaling.

1 Scale using MC30.

2 Scale using MC77 in the infinity norm.

3 Scale using MC77 in the one norm.

$\geq$4 Scale using MC64.

The default is scaling = 2.

small_dp is a scalar of type REAL(dp). Any pivot with absolute value less than small in a double precision factorization is treated as zero. The default is small_dp = epsilon(small_dp).

small_sp is a scalar of type REAL(sp). Any pivot with absolute value less than small in a single precision factorization is treated as zero. The default is small_sp = epsilon(small_sp).

u is a scalar of type REAL(dp) that is used by MA57 and HSL_MA77 for threshold partial pivoting. The default is u = 0.01. Values outside the range [0,0.5] are treated as the default.

Controls used only by MA79_refactor_solve

old_scaling is a scalar of type LOGICAL. If set to .true. and a scaling was previously computed on a call to MA79_factor_solve, this scaling will be reused (and control%scaling will be ignored). The default is old_scaling=.false..

Controls used by all routines (except MA77_finalize**)**

fgmres_init_rs_gap is a scalar of type INTEGER(short) that controls the initial gap between restarts within the FGMRES algorithm. The default is fgmres_init_rs_gap = 4. The gap between restarts is doubled if no progress is made in an iteration, up to a maximum of fgmres_max_rs_gap.
If fgmres_init_rs_gap lies outside the range [1, fgmres_max_rs_gap] it is taken as the nearest value in that range.

fgmres_max_itr is a scalar of type INTEGER(short) that controls the maximum total number of iterations of FGMRES. The default is fgmres_max_itr = 32. Values less than 0 are treated as 0, i.e. no FGMRES is performed.

fgmres_max_rs_gap is a scalar of type INTEGER(short) that controls the maximum gap between restarts within the FGMRES algorithm. The default is fgmres_max_rs_gap = 16. Values less than 1 are treated as 1.

fgmres_min_improve is a scalar of type REAL(dp) that holds the minimum improvement necessary for FGMRES with a given restart parameter to continue (the new residual must be at most fgmres_min_improve times the old residual otherwise restart will be doubled and/or FGMRES will terminate). The default is fgmres_min_improve = 0.3.

itref_max_itr is a scalar of type INTEGER(short) that controls the maximum number of iterations of FGMRES. The default is fgmres_max_itr = 10. Values less than 0 are treated as 0, i.e. no iterative refinement is performed.

itref_min_improve is a scalar of type REAL(dp) that holds the minimum improvement necessary for iterative refinement to continue (the new residual must be at most itref_min_improve times the old residual otherwise iterative refinement will terminate and FGMRES will be used). The default is itref_min_improve = 0.3.

use_ep is a scalar of type LOGICAL. If use_ep=.true., factors resulting from a single precision factorization are converted to double precision for the solves involved in FGMRES. This helps convergence and generally reduces the iteration count at the cost of each iteration being more expensive. The default is use_ep = .true..

2.6.9 The derived data type for returning information

The derived data type MA79_info is used to hold information about the progress and needs of the algorithm. This information will be incomplete if an error is returned. The components of MA79_info (in alphabetical order) are:

err_info is an array of type INTEGER(short) and size 3. It contains additional information on some error returns, see Section 2.7.

fgmres_itr is a scalar of type INTEGER(short). On exit from all routines involving a solve, it contains the total number of inner iterations (ie solves) used by FGMRES.

flag is a scalar of type INTEGER(short) that gives the exit status of the routine (details in Section 2.7).

itref_itr is a scalar of type INTEGER(short). On exit from all routines involving a solve, it contains the total number of iterations (ie solves) used by iterative refinement.

ma57_icntl is an array of type INTEGER(short) and size 20. If info%solver = 1, it holds the integer control parameters used by MA57. Otherwise, it holds the default settings for these parameters.

ma57_info is an array of type INTEGER(short) and size 40. If info%solver = 1, it contains the information values returned by the most recent calls to the MA57 routines. Otherwise, it is set to 0.

ma57d_cntl is an array of type REAL(dp) and size 20. If info%solver = 1 and info%factor_prec = 2, it contains the control parameters used by the double precision version of MA57. Otherwise, it holds the default settings for these parameters.

ma57d_info is an array of type REAL(dp) and size 20. If info%solver = 1 and info%factor_prec = 2, it contains the information values returned by the most recent call to the double precision MA57 routine. Otherwise, it is set to 0.

ma57_lfact is a scalar of type INTEGER(short). If info%solver = 1, it holds the size of the rank-one REAL(dp) array used by MA57 to hold the factors. Otherwise, it is set to 0.

ma57_lifact is a scalar of type INTEGER(short). If info%solver = 1, it holds the size of the rank-one INTEGER(short) array used by MA57 to hold the factors. Otherwise, it is set to 0.

ma57s_cntl is an array of type REAL(sp) and size 20. If info%solver = 1 and info%factor_prec = 1, it contains the control parameters used by the double precision version of MA57. Otherwise, it holds the default settings for these parameters.

ma57s_info is an array of type REAL(sp) and size 20. If info%solver = 1 and info%factor_prec = 1, it contains the information values returned by the most recent call to the single precision version of MA57. Otherwise, it is set to 0.

ma77s_cntl is a scalar of type MA77_control. If info%solver = 2 or 3 and info%factor_prec = 1, it contains the control parameters used by the single precision version of HSL_MA77. Otherwise, it holds the default settings for these parameters.

ma77s_info is a scalar of type MA77_info. If info%solver = 2 or 3 and info%factor_prec = 1, it contains the information values returned by the most recent call to the single precision version of HSL_MA77. Otherwise, the components are set to 0.

ma77d_cntl is a scalar of type MA77_control. If info%solver = 2 or 3 and info%factor_prec = 2, it contains the control parameters used by the single precision version of HSL_MA77. Otherwise, it holds the default settings for these parameters.

ma77d_info is a scalar of type MA77_info. If info%solver = 2 or 3 and info%factor_prec = 2, it contains the information values returned by the most recent call to the double precision version of HSL_MA77. Otherwise, the components are set to 0.

maxfront is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it holds the maximum front size during the factorization.

ndelay is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it holds the number of eliminations that were delayed.

nneg is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it holds the number of negative pivots chosen during the factorization.

normA is a scalar of type REAL(dp). If the optional argument sres_sub is not supplied then on exit from any routine involving a solve, info%normA holds the infinity norm of the matrix $\|\mathbf{A}\|_\infty = \max_i \sum_j |\mathbf{A}_{ij}|$.

ntwo is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it holds the number of 2×2 pivots chosen during the factorization.

prec is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it is set to 1 (respectively, 2) if the last factorization to be performed used single (respectively, double) precision arithmetic.

psdef is a scalar of type LOGICAL. If a positive definite factorization was used posdef will be set to .true., otherwise it is set to .false..

solver is a scalar of type INTEGER(short). On exit from MA79_factor_solve or MA79_refactor_solve, it is set to 1 (respectively, 2) if ma57 (respectively, HSL_MA77) was chosen and used successfully; it is set to 3 if the factorization initially used MA57 but a switch was made to HSL_MA77 because there was insufficient memory to allocate all the arrays needed by MA57 (this can only happen in the indefinite case).

timings is an array of type REAL(sp) and size 8. If control%time_parts=.true. then after each routine the relevant components of timings contain in the time for that part of the solution. Other components will have the value -1. Times are in seconds and are limited in resolution by the accuracy of calls to the Fortran intrinsic function system_clock.

- timings(1) Time to calculate scaling of matrix.
- timings(2) Time to perform analyse using MA57.
- timings(3) Time to perform analyse using HSL_MA77.
- timings(4) Time to input any new matrix data into HSL_MA77.
- timings(5) Time to perform the most recent factorization phase.
- timings(6) Time to perform the most recent call to MA57C/CD or MA77_solve. (Other solves may be faster or slower due to caching effects).
- timings(7) Time for iterative refinement.
- timings(8) Time for FGMRES.

warnings is a scalar of type INTEGER(short) that holds a count of the number of warnings encountered in the last routine called. Multiple warnings of the same type are all counted.

2.7 Warning and error messages

A successful return from a subroutine in the package is indicated by info%flag having a zero value. A negative value indicates an error value while a positive value indicates a warning. The meanings of these values are shown below:

-1 A%n is out of range.

-2 A%ptr is unallocated or has size less than A%n+1.

-3 A%ptr has an out of range value or a non-monotonic increase at index info%err_info(1).

-4 A%row is unallocated or is of size insufficient to accommodate both lower and upper entries of the matrix A.

-5 A%val is unallocated or has size insufficient to accomodate both lower and upper entires of the matrix A.

-10 nrhs<0, lrhs<A%n or size of array rhs does not match these values.

-11 order has size less than A%n.

-12 order is not a permutation. The first repeated index is at position err_info(1)

-13 order has an out of range value at position err_info(1).

-14 order has unmatched 2×2 pivots at positions err_info(1) and err_info(2).

-15 resid has size less than nrhs.

-20 control%bits is out of range.

-21 control%maxmem is out of range.

-22 control%prec is out of range.

-23 control%solver is out of range.

-24 control%fgmres_init_rs_gap is out of range.

-25 control%fgmres_max_rs_gap is out of range.

-30 Error in allocating memory. The Fortran stat value is available in info%err_info(1).

-31 Error in deallocating memory. The Fortran stat value is available in info%err_info(1).

-32 Calls to HSL_MA79 out of sequence.

-33 Code attempted to use HSL_MA77 but files specified in control%ma77_files already exist in path.

-34 psdef=.true. but supplied problem was found not to be positive definite (control%action_indef=.false.).

-35 Matrix is singular (control%action_singular=.false.).

-100 Error return from MA57. The MA57 return code is in err_info(1).

-101 Error return from HSL_MA77. The HSL_MA77 return code is in err_info(1).

+1 accuracy is very small and could have caused excessive iterations to reach convergence.

+2 Out of range entry in A%row at position err_info(1); entry was ignored. Only returned by MA79_factor_solve.

+3 Duplicate entries in A at positions err_info(1) and err_info(2); entries were summed. Only returned by MA79_factor_solve.

+4 control%solver=1 (use MA57) but insufficient memory for MA57 so HSL_MA77 was used instead.

+5 MC64 produced large scaling factors and the resultant factorization may not be accurate.

+6 psdef=.true. but supplied problem was found not to be positive definite (control%action_indef=.true.).

+7 Matrix is singular (control%action_singular=.true.).

+8 Failed to reach desired accuracy after attempting all allowable fallback options.

Note: Warning flags will overwrite previous warnings, a count of warnings is kept in info%warnings.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: HSL_MA54, MA57, HSL_MA64, HSL_MA77, MC30, MC34, MC64, MC77, MI15, HSL_OF01, HSL_ZD11.

Input/output: Output is provided under the control of control%print_level, which allows error, warning and diagnostics messages to be printed on the relevant units control%unit_error, control%unit_warning and control%unit_diagnostics. Printing from called codes is available when diagnostics are enabled. I/O to direct-access files whose unit numbers are chosen by HSL_OF01 may result from the use of HSL_MA77.

Restrictions: A%n>0, nrhs≥0, lrhs≥A%n, bits=32 or 64, control%solver=0,1 or 2, control%ma57_blocking>0, control%ma77_nb>0, control%prec=1 or 2, if control%bits=32 then control%maxmem<huge(0_short)/8.

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

4 METHOD

Algorithm 1 Basic outline of mixed precision solver

```
    Check user-supplied data for errors
    Perform MA57 analyse phase
    Choose solver; initialise prec
    loop
5:    Factorise A using solver in precision prec.
      Use the computed factors to solve Ax = b using solve phase of solver in double precision.
      Compute scaled residual β
      if β ≤ accuracy then return
      Apply iterative refinement (Algorithm 2). Recompute β.
10:   if β ≤ accuracy then return
      Apply FGMRES (Algorithm 3). Recompute β.
      if β ≤ accuracy then return
      if prec = SINGLE and fallback_double then
        prec = DOUBLE
15:   else
        return with warning that accuracy not achieved.
      end if
    end loop
```

Algorithm 2 Iterative Refinement

```
    Input: Computed factors, accuracy, itref_max_itr, itref_min_reduce
    Solve Ax^(1) = b and set r^(1) = b − Ax^(1)
    Set k = 1 and compute scaled residual β^(1).
    while β^(k) >accuracy and k ≤ itref_max_itr do
5:    Solve Ax^(k) = r^(k)
      Set x^(k+1) ← x^(k) + x^(k+1)
      Compute r^(k+1) = b − Ax^(k+1) and β^(k+1)
      if itref_min_reduce×β^(k+1) > β^(k) then exit {Insufficient Reduction}
      k = k + 1
10: end while
    x = x^(k) : k = arg min^(k) β^(k)
    β = min^(k) β^(k)
```

The method implemented for the mixed precision solver is outlined in Algorithm 1, corresponding to a call to MA79_factor_solve. Calls to MA79_refactor_solve correspond to this algorithm entering at line 4 (the start of the loop). Calls to MA79_solve correspond to lines 6–12 only.

The two iterative methods available for recovering precision are detailed with their major control parameters as Algorithms 2 and 3, shown for a single right hand side only for clarity, and $\beta^{(k)}$ corresponds to the scaled residual for the kth iteration.

A full technical report covering these algorithms is available as [1].

Reference:

[1] J.D. Hogg and J.A. Scott. (2008). A fast and robust mixed precision solver for the solution of sparse symmetric linear systems. Technical Report TR-RAL-2008-23

Algorithm 3 FGMRES with adaptive restarting

Input: Computed factors, `accuracy`, `fgmres_max_itr`, `fgmres_init_rs_gap`, `fgmres_max_rs_gap`, `fgmres_min_improve`
Solve $\mathbf{Ax} = \mathbf{b}$. Set $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$. Calculate scaled residual $\mathbf{r}$
Initialise `MI15`. Set *total_itr*= 0, *restart*=`fgmres_init_rs_gap`.
Set *prev_res*=`huge(0.0)`.
while $\beta >$`accuracy` **and** *total_itr* $<$ `fgmres_max_itr` **do**
  if $\beta \geq$`fgmres_min_improve`$\times$*prev_res* **then**
    restart= $2\times$*restart*
    if *restart*$>$`fgmres_max_rs_gap` **then** fail.
    Reinitialize `MI15`.
  end if
  if $\beta >$*prev_res* **then**
    Restore previous solution.
  else
    Store new solution; Set *prev_res*= β.
  end if
  Call `MI15` to perform one outer iteration of right-preconditioned FGMRES(restart) to update $\mathbf{x}$.
  Calculate new $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ and β.
end while

5 EXAMPLE OF USE

5.1 Basic usage

We give an example of the code required to solve a set of equations using `HSL_MA79` demonstrating calls to `MA79_factor_solve` and `MA79_refactor_solve`.
Consider we wish to solve the system

$$\begin{pmatrix} 1.00 & 0.86 & & 1.23 \\ 0.86 & 1.00 & & \\ & & 2.50 & 3.10 \\ 1.23 & & 3.10 & 4.00 \end{pmatrix} \mathbf{X} = \begin{pmatrix} 3.09 & 2.045 \\ 1.86 & 1.36 \\ 5.60 & 4.05 \\ 8.33 & 6.33 \end{pmatrix}$$

and then the system

$$\begin{pmatrix} 2.00 & 1.72 & & 2.46 \\ 1.72 & 2.00 & & \\ & & 5.00 & 6.20 \\ 2.46 & & 6.20 & 8.00 \end{pmatrix} \mathbf{X} = \begin{pmatrix} 3.09 \\ 1.86 \\ 5.60 \\ 8.33 \end{pmatrix}$$

We use `MA79_factor_solve` and `MA79_refactor_solve` as follows:

```
! Simple code to illustrate use of hsl_ma79
program hsl_ma79ds
   use hsl_ma79_double
   use hsl_zd11_double
   implicit none
   integer, parameter :: dp = kind(0d0)

   ! Derived types for HSL_MA79
```

```
type(ma79_keep) :: keep
type(ma79_control) :: control
type(ma79_info) :: info

! Input/Output Variables
type(zd11_type) :: matrix
real(kind=dp), dimension(:, :), allocatable :: rhs
real(kind=dp), dimension(:), allocatable :: resid
logical :: posdef
real(kind=dp) :: accuracy
integer :: nrhs

! Local Variables
integer :: i, j

!
! Read in lower triangular part of matrix in following format:
! n ne
! ptr(1:n+1)
! row(1:ne)
! val(1:ne)
!
read (*,*) matrix%n, matrix%ne
allocate(matrix%ptr(matrix%n+1))
allocate(matrix%row(2*matrix%ne), matrix%col(2*matrix%ne))
allocate(matrix%val(2*matrix%ne))

read (*,"(5i8)") (matrix%ptr(i), i=1,matrix%n+1)
read (*,"(5i8)") (matrix%row(i), i=1,matrix%ne)
read (*,"(5es12.4)") (matrix%val(i), i=1,matrix%ne)

!
! Read in right hand side(s)
!
read(*, *) nrhs
allocate(rhs(matrix%n, nrhs), resid(nrhs))
do i = 1, nrhs
   read (*,"(5es12.4)") (rhs(j, i), j=1,matrix%n)
end do

!
! Call MA79 to perform factorization and solve
!
posdef = .false.
accuracy = 1d-14
call ma79_factor_solve(matrix, posdef, nrhs, matrix%n, rhs, accuracy, keep, &
   control, info, resid=resid)
print "(a, 5es12.4)", "ma79_factor_solve residuals are ", resid(:)
print "(a, 5es12.4)", "ma79_factor_solve soln(1) = ", rhs(:, 1)
```

```
   print "(a, 5es12.4)", "ma79_factor_solve soln(2) = ", rhs(:, 2)

   !
   ! Read in some new numerical data. Note that now we must address a full
   ! matrix returned from ma79 rather than a lower triangular one. We expect
   ! the data we read to be ordered correctly (i.e. by row indices within
   ! columns).
   !
   read (*,"(5es12.4)") (matrix%val(i), i=1,matrix%ne)
   nrhs = 1
   read (*,"(5es12.4)") (rhs(i, 1), i=1,matrix%n)

   !
   ! Call MA79 to refactorize and then solve for a single rhs
   !
   call ma79_refactor_solve(matrix, posdef, nrhs, matrix%n, rhs, accuracy, &
      keep, control, info, resid=resid)
   print "(/, a, es12.4)", "ma79_refactor_solve residual is ", resid(1)
   print "(a, 5es12.4)", "ma79_refactor_solve soln(1) = ", rhs(:, 1)

   !
   ! Cleanup after MA79 (closes files etc)
   !
   call ma79_finalize(keep, control, info)
end program hsl_ma79ds
```

The input file is:

```
      4       7
      1       4       5       7       8
      1       2       4       2       3
      4       4
    1.0E+00     8.6E-01    1.23E+00     1.0E+00     2.5E+00
    3.1E+00     4.0E+00
      2
   3.09E+00    1.86E+00    5.60E+00    8.33E+00
  2.045E+00    1.36E+00    4.05E+00    6.33E+00
    2.0E+00    1.72E+00    2.46E+00    1.72E+00     2.0E+00
    5.0E+00     6.2E+00    2.46E+00     6.2E+00     8.0E+00
   3.09E+00    1.86E+00    5.60E+00    8.33E+00
```

This produces the following output:

```
ma79_factor_solve residuals are   6.9572E-15  9.0045E-15
ma79_factor_solve soln(1) =   1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
ma79_factor_solve soln(2) =   1.0000E+00  5.0000E-01  1.0000E+00  5.0000E-01

ma79_refactor_solve residual is   8.9831E-15
ma79_refactor_solve soln(1) =   5.0000E-01  5.0000E-01  5.0000E-01  5.0000E-01
```

5.2 Use of a user supplied norm

Consider solving the first system given above but using a component-wise norm

$$\begin{aligned} \beta_{\mathbf{i}} &= \max_j \frac{|(A\mathbf{x_i} - \mathbf{b_i})_j|}{\gamma_{ij}} \\ \gamma_{ij} &= \begin{cases} |A_j||x_{ij}| + |b_{ij}| & |A_j||x_{ij}| + |b_{ij}| \geq \varepsilon \\ 1 & |A_j||x_{ij}| + |b_{ij}| < \varepsilon \end{cases} \end{aligned}$$

where ε is machine precision. We implement this as the subroutine `cwnorm`, and use it with `HSL_MA79` as follows:

```
! Simple code to illustrate use of sres_sub
program hsl_ma79ds1
   use hsl_ma79_double
   use hsl_zd11_double
   implicit none

   integer, parameter :: dp = kind(0d0)

   ! Derived types for HSL_MA79
   type(ma79_keep) :: keep
   type(ma79_control) :: control
   type(ma79_info) :: info

   ! Input/Output Variables
   type(zd11_type) :: matrix
   real(kind=dp), dimension(:, :), allocatable :: rhs
   real(kind=dp), dimension(:), allocatable :: resid
   logical :: posdef
   real(kind=dp) :: accuracy
   integer :: nrhs

   ! Local Variables
   integer :: i, j

   interface
      subroutine cwnorm(A, x, b, r, retain, res)
         use HSL_ZD11_double
         implicit none
         integer, parameter :: dp = kind(0d0)

         type(ZD11_type), intent(in) :: A
         real(kind=dp), dimension(:, :), intent(in) :: x
         real(kind=dp), dimension(:, :), intent(in) :: b
         real(kind=dp), dimension(:, :), intent(in) :: r
         real(kind=dp), dimension(:), allocatable, intent(inout) :: retain
         real(kind=dp), dimension(:), intent(out) :: res
      end subroutine
   end interface
```

```
   !
   ! Read in lower triangular part of matrix:
   read (*,*) matrix%n, matrix%ne
   allocate(matrix%ptr(matrix%n+1))
   allocate(matrix%row(2*matrix%ne), matrix%col(2*matrix%ne))
   allocate(matrix%val(2*matrix%ne))

   read (*,"(5i8)") (matrix%ptr(i), i=1,matrix%n+1)
   read (*,"(5i8)") (matrix%row(i), i=1,matrix%ne)
   read (*,"(5es12.4)") (matrix%val(i), i=1,matrix%ne)

   !
   ! Read in right hand side(s)
   !
   read(*, *) nrhs
   allocate(rhs(matrix%n, nrhs), resid(nrhs))
   do i = 1, nrhs
      read (*,"(5es12.4)") (rhs(j, i), j=1,matrix%n)
   end do

   !
   ! Call MA79 to perform factorization and solve
   !
   posdef = .false.
   accuracy = 1d-14
   call ma79_factor_solve(matrix, posdef, nrhs, matrix%n, rhs, accuracy, keep, &
      control, info, resid=resid, sres_sub=cwnorm)
   print "(a, 5es12.4)", "Residuals are ", resid(:)
   print "(a, 5es12.4)", "soln(1) = ", rhs(:, 1)
   print "(a, 5es12.4)", "soln(2) = ", rhs(:, 2)

   !
   ! Cleanup after MA79 (closes files etc)
   !
   call ma79_finalize(keep, control, info)
end program hsl_ma79ds1

subroutine cwnorm(A, x, b, r, retain, res)
   use HSL_ZD11_double
   implicit none
   integer, parameter :: dp = kind(0d0)

   type(ZD11_type), intent(in) :: A
   real(kind=dp), dimension(:, :), intent(in) :: x
   real(kind=dp), dimension(:, :), intent(in) :: b
   real(kind=dp), dimension(:, :), intent(in) :: r
   real(kind=dp), dimension(:), allocatable, intent(inout) :: retain
   real(kind=dp), dimension(:), intent(out) :: res
```

```
   integer :: i, j
   real(kind=dp), dimension(size(res)) :: temp

   if(.not.allocated(retain)) then
      allocate(retain(A%n))
      do i = 1, A%n
         retain(i) = 0
         do j = A%ptr(i), A%ptr(i+1)-1
            retain(i) = retain(i) + abs(A%val(j))
         end do
      end do
   endif

   res(:) = 0
   do i = 1, A%n
      temp(:) = (retain(i) * abs(x(i, :)) + abs(b(i, :)))
      where(temp<epsilon(temp))
         res = max(res, abs(r(i, :)))
      elsewhere
         res = max(res, abs(r(i, :)) / temp)
      endwhere
   end do
end subroutine cwnorm
```

The input file is:

```
       4       7
       1       4       5       7       8
       1       2       4       2       3
       4       4
     1.0E+00     8.6E-01    1.23E+00     1.0E+00     2.5E+00
     3.1E+00     4.0E+00
       2
    3.09E+00    1.86E+00    5.60E+00    8.33E+00
   2.045E+00    1.36E+00    4.05E+00    6.33E+00
```

This produces the following output:

```
Residuals are   3.9651E-17  4.8481E-17
soln(1) =   1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
soln(2) =   1.0000E+00  5.0000E-01  1.0000E+00  5.0000E-01
```