# GALAHAD ULS

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

## 1 SUMMARY

This package **solves dense or sparse unsymmetric systems of linear equations** using variants of Gaussian elimination. Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$, and an $m$-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{A}\mathbf{x} = \mathbf{b}$. If $\mathbf{b}$ is an $n$-vector, the subroutine may solve instead the system $\mathbf{A}^T\mathbf{x} = \mathbf{b}$. Both square ($m = n$) and rectangular ($m \neq n$) matrices are handled; one of an infinite class of solutions for consistent systems will be returned whenever $\mathbf{A}$ is not of full rank.

The method provides a common interface to a variety of well-known solvers from HSL and elsewhere. Currently supported solvers include `MA28/GLS` and `HSL_MA48`, as well as `GETR` from LAPACK. Note that **the solvers themselves do not form part of this package and must be obtained separately.** Dummy instances are provided for solvers that are unavailable. Also note that additional flexibility may be obtained by calling the solvers directly rather that via this package.

**ATTRIBUTES — Versions:** `GALAHAD_ULS_single`, `GALAHAD_ULS_double`. **Calls:** `GALAHAD_SYMBOLS`, `GALAHAD_SORT`, `GALAHAD_SPACE`, `GALAHAD_SPECFILE`, `GALAHAD_STRING`, `GALAHAD_SMT`, `GALAHAD_GLS` and optionally `HSL_MA48`. **Date:** August 2009. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Parallelism:** Some solvers may use OpenMP and its runtime library.

## 2 HOW TO USE THE PACKAGE

The package is available using both single and double precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_ULS_single
```

with the obvious substitution `GALAHAD_ULS_double`, `GALAHAD_ULS_single_64` and `GALAHAD_ULS_double_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `SMT_type`, `ULS_control_type`, `ULS_data_type`, and `ULS_inform_type` (§2.5), and the subroutines `ULS_initialize`, `ULS_factorize`, `ULS_solve`, `ULS_terminate` (§2.6), and `ULS_enquire` (§2.8) must be renamed on one of the `USE` statements.

There are four principal subroutines for user calls (see §2.8 for further features):

The subroutine `ULS_initialize` must be called to specify the external solver to be used. It may also be called to set default values for solver-specific components of the control structure. If non-default values are wanted for any of the control components, the corresponding components should be altered after the call to `ULS_initialize`.

`ULS_factorize` accepts the pattern of $\mathbf{A}$ and chooses pivots for Gaussian elimination using a selection criterion to preserve sparsity. The factors of $\mathbf{A}$ are generated using the calculated pivot order.

`ULS_solve` uses the factors generated by `ULS_factorize` to solve a system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{x} = \mathbf{b}$. Iterative refinement may be used to improve a given solution or set of solutions.

`ULS_terminate` deallocates the arrays held inside the structure for the factors. It should be called when all the systems involving its matrix have been solved or before another external solver is to be used.

### 2.1 Supported external solvers

In Table 2.1 we summarize key features of the external solvers supported by ULS. Further details are provided in the references cited in §4.

| solver | factorization | out-of-core | parallelised |
|---|---|---|---|
| GLS/MA28 | sparse | no | no |
| HSL_MA48 | sparse | no | no |
| GETR | dense | no | with parallel LAPACK |

Table 2.1: External solver characteristics.

### 2.2 Matrix storage formats

The matrix $\mathbf{A}$ may be stored in a variety of input formats.

#### 2.2.1 Sparse co-ordinate storage format

Only the nonzero entries of $\mathbf{A}$ are stored. For the $l$-th entry of $\mathbf{A}$, its row index $i$, column index $j$ and value $a_{ij}$ are stored in the $l$-th components of the integer arrays row, col and real array val, respectively. The order is unimportant, but the total number of entries ne is also required.

#### 2.2.2 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row $i$ appear directly before those in row $i+1$. For the $i$-th row of $\mathbf{A}$, the $i$-th component of an integer array ptr holds the position of the first entry in this row, while ptr $(m+1)$ holds the total number of entries plus one. The column indices $j$ and values $a_{ij}$ of the entries in the $i$-th row are stored in components $l =$ ptr$(i)$, ...,ptr $(i+1)-1$ of the integer array col, and real array val, respectively.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

#### 2.2.3 Dense storage format

The matrix $\mathbf{A}$ is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. In particular, component $m*(i-1)+j$ of the storage array val will hold the value $a_{ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

### 2.3 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords REAL(rp_) and INTEGER(ip_), where rp_ and ip_ are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default REAL for the single precision versions and DOUBLE PRECISION for the double precision cases, and correspond to rp_ = real32 and rp_ = real64, respectively, as supplied by the fortran iso_fortran_env module. The latter are default (32-bit) and long (64-bit) integers, and correspond to ip_ = int32 and ip_ = int64, respectively, again from the iso_fortran_env module.

### 2.4 Parallel usage

OpenMP may be used by the GALAHAD_ULS package to provide parallelism for some solvers in shared memory environments. See the documentation for the GALAHAD package SLS for more details. 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.
MPI may also be used by the package to provide parallelism for some solvers in a distributed memory environment. To use this form of parallelism, MPI must be enabled at runtime by using the correct compiler flag (usually some variant of -lmpi). Although the MPI process will be started automatically when required, it should be stopped by the calling program once no further use of this form of parallelism is needed. Typically, this will be via statements of the form

```
CALL MPI_INITIALIZED( flag, ierr )
IF ( flag ) CALL MPI_FINALIZE( ierr )
```

The code may be compiled and run in serial mode.

### 2.5 The derived data types

Four derived data types are used by the package.

#### 2.5.1 The derived data type for holding the matrix

The derived data type SMT_type is used to hold the matrix **A**. The components of SMT_type used are:

m is a scalar variable of type INTEGER(ip_), that holds the row dimension $m$ of the matrix **A**. **Restriction:** m $\geq$ 1.

n is a scalar variable of type INTEGER(ip_), that holds the column dimension $n$ of the matrix **A**. **Restriction:** n $\geq$ 1.

type is an allocatable array of rank one and type default CHARACTER, that indicates the storage scheme used. If the sparse co-ordinate scheme (see §2.2.1) is used the first ten components of type must contain the string COORDINATE. For the sparse row-wise storage scheme (see §2.2.2), the first fourteen components of type must contain the string SPARSE_BY_ROWS, and for dense storage scheme (see §2.2.3) the first five components of type must contain the string DENSE.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into type. For example, if **A** is to be stored in the structure A of derived type SMT_type and we wish to use the co-ordinate scheme, we may simply

```
CALL SMT_put( A%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

ne is a scalar variable of type INTEGER(ip_), that holds the number of entries in **A** in the sparse co-ordinate storage scheme (see §2.2.1). It need not be set for any of the other three schemes.

val is a rank-one allocatable array of type REAL(rp_), that holds the values of the entries of the matrix **A** for each of the storage schemes discussed in §2.2. Any duplicated entries that appear in the sparse co-ordinate or row-wise schemes will be summed.

row is a rank-one allocatable array of type INTEGER(ip_), that holds the row indices of **A** in the sparse co-ordinate storage scheme (see §2.2.1). It need not be allocated for any of the other schemes. Any entry whose row index lies out of the range [1,n] will be ignored.

col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of **A** in either the sparse co-ordinate (see §2.2.1), or the sparse row-wise (see §2.2.2) storage scheme. It need not be allocated when the dense storage scheme is used. Any entry whose row index lies out of the range [1,m] or column index lies out of the range [1,n] will be ignored.

ptr is a rank-one allocatable array of size m+1 and type INTEGER(ip_), that holds the starting position of **A**, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see §2.2.2). It need not be allocated for the other schemes.

Although any of the above-mentioned matrix storage formats may be used with each supported solver, MA28/GLS and HSL_MA48 from HSL are most efficient if co-ordinate input is provided.

### 2.5.2 The derived data type for holding control parameters

The derived data type ULS_control_type is used to hold controlling data. Default values specifically for the desired solver may be obtained by calling ULS_initialize (see §2.6.1), while components may be changed at run time by calling ULS_read_specfile (see §2.9.1). The components of ULS_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the unit number for error messages. Printing of error messages is suppressed if error $< 0$. The default is error = 6.

warning is a scalar variable of type INTEGER(ip_), that holds the unit number for warning messages. Printing of warning messages is suppressed if warning $< 0$. The default is warning = 6.

out is a scalar variable of type INTEGER(ip_), that holds the unit number for informational messages. Printing of informational messages is suppressed if out $< 0$. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output that is required. No informational output will occur if print_level $\leq 0$. If print_level $\geq 1$ a single line of output will be produced for each step of iterative refinement performed. If print_level $\geq 2$ this output may be increased to provide significant detail of the factorization process. The default is print_level = 0.

print_level_solver is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output that is required by the exteral solver. No informational output will occur if print_level $\leq 0$. If print_level $\geq 1$ the amount of output produced is solver dependent. The default is print_level_solver = 0.

initial_fill_in_factor is a scalar variable of type INTEGER(ip_), that gives a prediction of the factor by which the fill-in will exceed the initial number of entries in **a**. The default is initial_fill_in_factor = 3.

min_real_factor_size is a scalar variable of type INTEGER(ip_), that specifies the amount of real storage that will initially be allocated for the factors and other data. The default is min_real_factor_size = 10000, and this default is used if min_real_factor_size < 1.

min_integer_factor_size is a scalar variable of type INTEGER(ip_), that specifies the amount of integer storage that will initially be allocated for the factors and other data. The default is min_integer_factor_size = 10000, and this default is used if min_integer_factor_size < 1.

max_factor_size is a scalar variable of type INTEGER(int64), that specifies the maximum amount of real storage that will be allocated for the factors and other data. The default is max_factor_size = HUGE(0).

blas_block_size_factorize is a scalar variable of type INTEGER(ip_), that gives the block size for level-three basic linear algebra subprograms (BLAS) in the factorization phase. The default is blas_block_size_factorize = 16, and this default is used if blas_block_size_factorize < 1.

blas_block_size_solve is a scalar variable of type INTEGER(ip_), that gives the block size for level-two and -three basic linear algebra subprograms (BLAS) in the solution phase. The default is blas_block_size_solve = 16, and this default is used if blas_block_size_solve < 1.

pivot_control is a scalar variable of type INTEGER(ip_), that is used to control numerical pivoting by ULS_factorize. Possible values are:

1 Threshold partial pivoting will be performed, with relative pivot tolerance given by the component relative_pivot_tolerance.

2 Threshold rook pivoting is desired, with relative pivot tolerance given by the component relative_pivot_tolerance.

3 Threshold complete pivoting is desired, with relative pivot tolerance given by the component relative_pivot_tolerance.

4 Threshold symmetric pivoting is desired, with relative pivot tolerance given by the component relative_pivot_tolerance.

5 Threshold diagonal pivoting is desired, with relative pivot tolerance given by the component relative_pivot_tolerance.

The default is pivot_control = 1, and any value outside of $[1,5]$ will be reset to the default. If a desired value is not available, the default will be substituted.

pivot_search_limit is a scalar variable of type INTEGER(ip_), that controls the maximum number of row and columns searched for a pivot. If pivot_search_limit $\leq$ 0, the search is unlimited. The default is pivot_search_limit = 0.

minimum_size_for_btf is a scalar variable of type INTEGER(ip_), that specifies the minimum size of a block within any block-triangular form found during the factorization. The default is minimum_size_for_btf = 1.

max_iterative_refinements is a scalar variable of type default INTEGER(ip_), that holds the maximum number of iterative refinements that may be attempted. The default is max_iterative_refinements = 0.

stop_if_singular is a scalar variable of type default LOGICAL, that should be set .TRUE. if the factorization is to be terminated if **A** is found to be singular, and .FALSE. if the factorization should continue. The default is stop_if_singular = .FALSE..

array_increase_factor is a scalar variable of type REAL(rp_), that holds the factor by which arrays sizes are to be increased if they are too small. The default is array_increase_factor = 2.0.

switch_to_full_code_density is a scalar variable of type REAL(rp_), that specifies the density at which a switch to full/dense code to perform the remaining factorization occurs. The default is switch_to_full_code_density = 0.5.

array_decrease_factor is a scalar variable of type REAL(rp_), that holds a factor which is used to assess whether previously allocated internal workspace arrays are excessive. In particular, if current requirements are less than array_decrease_factor times the currently allocated space, the space will be re-allocated to current requirements. The default is array_decrease_factor = 2.0.

relative_pivot_tolerance is a scalar variable of type REAL(rp_), that holds the relative pivot tolerance that is used to control the stability of the factorization. The default is relative_pivot_tolerance = 0.01. For problems requiring greater than average numerical care a higher value than the default would be advisable. Values greater than 0.5 are treated as 0.5 and less than 0.0 as 0.0.

absolute_pivot_tolerance is a scalar variable of type REAL(rp_), that holds the absolute pivot tolerance which is used to control the stability of the factorization. No pivot smaller than absolute_pivot_tolerance in absolute value will be accepted. The default is absolute_pivot_tolerance = EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_ULS_double).

zero_tolerance is a scalar variable of type REAL(rp_), that controls which small entries are to be ignored during the factorization of $\mathbf{A}$. Any entry smaller in absolute value than zero_tolerance will be treated as zero; as a consequence when zero_tolerance > 0, the factors produced will be of a perturbation of order zero_tolerance of $\mathbf{A}$. The default is zero_tolerance = 0.0.

acceptable_residual_relative and acceptable_residual_absolute are scalar variables of type REAL(rp_), that specify an acceptable level for the residual $\mathbf{A}\mathbf{x}-\mathbf{b}$ or residuals $\mathbf{A}\mathbf{x}_i-\mathbf{b}_i$, $i=1,\ldots,r$, when there are more than one. In particular, iterative refinement will cease as soon as $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_\infty$ falls below $\max(\|\mathbf{b}\|_\infty *$ acceptable_residual_relative, acceptable_residual_absolute); for the multiple residual case, we require that $\|\mathbf{A}\mathbf{x}_i-\mathbf{b}_i\|_\infty$ falls below $\max(\|\mathbf{b}_i\|_\infty *$ acceptable_residual_relative, for each $i=1,\ldots,r$. The defaults are acceptable_residual_relative = acceptable_residual_absolute = $10u$, where $u$ is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_ULS_double).

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, the default prefix = "" should be used.

### 2.5.3 The derived data type for holding informational parameters

The derived data type ULS_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of ULS_inform_type are as follows—any component that is not relevant to the solver being used will have the value -1 or -1.0 as appropriate:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See §2.7 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or deallocation.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if there have been no allocation or deallocation errors.

out_of_range is a scalar variable of type INTEGER(int64), that is set to the number of entries of $\mathbf{A}$ supplied with one or both indices out of range.

duplicates is a scalar variable of type INTEGER(int64), that is set to the number of duplicate off-diagonal entries of $\mathbf{A}$ supplied.

entries_dropped is a scalar variable of type INTEGER(int64), that is set to the number of small entries dropped from the factorization.

workspace_factors is a scalar variable of type INTEGER(int64), that gives the total number of reals and integers needed to hold the factors.

compresses is a scalar variable of type INTEGER(ip_), that gives the number of compresses of the workspace data required.

entries_in_factors is a scalar variable of type INTEGER(int64), that gives the number of entries in the factors of $\mathbf{A}$.

rank is a scalar variable of type INTEGER(ip_), that gives an estimate of the rank of $\mathbf{A}$.

structural_rank is a scalar variable of type INTEGER(ip_), that gives the structural rank of $\mathbf{A}$.

pivot_control is a scalar variable of type INTEGER(ip_), that specifies what form of numerical pivoting has been used. Possible values are:

- 1 Threshold partial pivoting has been performed.
- 2 Threshold rook pivoting has been performed.
- 3 Threshold complete pivoting has been performed.
- 4 Threshold symmetric pivoting has been performed.
- 5 Threshold diagonal pivoting has been performed.

iterative_refinements is a scalar variable of type INTEGER(ip_), that gives the number of iterative refinements performed.

solver is a scalar variable of type default CHARACTER and length 20, that gives the name of the actual solver used.

gls_ainfo is a scalar variable of type gls_ainfo, that corresponds to the output value gls_ainfo from GLS. See the documentation for GLS for further details.

gls_finfo is a scalar variable of type gls_finfo, that corresponds to the output value gls_finfo from GLS. See the documentation for GLS for further details.

gls_sinfo is a scalar variable of type gls_sinfo, that corresponds to the output value gls_sinfo from GLS. See the documentation for GLS for further details.

ma48_ainfo is a scalar variable of type ma48_ainfo, that corresponds to the output value ma48_ainfo from HSL_MA48. See the documentation for HSL_MA48 for further details.

ma48_finfo is a scalar variable of type ma48_finfo, that corresponds to the output value ma48_finfo from HSL_MA48. See the documentation for HSL_MA48 for further details.

ma48_sinfo is a scalar variable of type ma48_sinfo, that corresponds to the output value ma48_sinfo from HSL_MA48. See the documentation for HSL_MA48 for further details.

PARDISO_error is a scalar variable of type INTEGER(ip_), that corresponds to the output value error from PARDISO. See the documentation for PARDISO for further details.

PARDISO_iparm is an array of size 64 and type INTEGER(ip_), whose components correspond to those in the output array IPARM from PARDISO. See the documentation for PARDISO for further details.

PARDISO_dparm is an array of size 64 and type REAL(rp_), whose components correspond to those in the output array DPARM from PARDISO. See the documentation for PARDISO for further details.

lapack_error is a scalar variable of type INTEGER(ip_), that corresponds to the output value info returned from the LAPACK routine S/DGETRF/S. See the documentation for LAPACK for further details.

#### 2.5.4 The derived data type for holding problem data

The derived data type ULS_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls to ULS procedures. All components are private.

### 2.6 Argument lists and calling sequences

#### 2.6.1 The initialization subroutine

The initialization subroutine must be called for each solver used to initialize data and solver-specific control parameters.

```
CALL ULS_initialize( solver, data, control, inform[, check] )
```

`solver` is scalar, of `INTENT(IN)`, of type `CHARACTER`, and of variable length that specifies which solver to use. Possible values are

`gls` if the GALAHAD solver `GLS` is desired.

`ma28` is an alias for `gls` that reflects the fact that the GALAHAD solver `GLS` is a Fortran-90 encapsulation of the Fortran-77 package `MA28` from HSL.

`ma48` if the HSL solver `HSL_MA48` is desired. This is a more advanced version of `GLS/MA28`.

`getr` if the LAPACK dense LU factorization package `(S/D)GETR(F/S)` is desired.

Other solvers may be added in the future.

`data` is a scalar `INTENT(OUT)` argument of type `ULS_data_type` (see §2.5.4). It is used to hold data about the problem being solved.

`control` is a scalar `INTENT(OUT)` argument of type `ULS_control_type` (see §2.5.2). On exit, `control` contains solver-specific default values for the components as described in §2.5.2. These values should only be changed after calling `ULS_initialize`.

`inform` is a scalar `INTENT(OUT)` argument of type `ULS_inform_type` (see §2.5.3). A successful call is indicated when the component `status` has the value 0. For other return values of `status`, see §2.7.

`check` is an `OPTIONAL` scalar `LOGICAL INTENT(IN)` argument that if `PRESENT` and set `.TRUE.` will check to see if the requested solver is available, and if not will replace this by a suitable equivalent; the equivalent will be recorded in `control%solver`. No checks will be performed if `check` is not `PRESENT` or if it is set to `.FALSE.` .

#### 2.6.2 The factorization subroutine

The matrix $\mathbf{A}$ may be factorized as follows:

```
CALL ULS_factorize( matrix, data, control, inform )
```

`matrix` is scalar `INTENT(IN)` argument of type `SMT_type` that is used to specify $\mathbf{A}$. The user must set all of the relevant components of `matrix` according to the storage scheme desired (see §2.5.1). Incorrectly-set components will result in errors flagged in `inform%status`, see §2.7.

`data` is a scalar `INTENT(INOUT)` argument of type `ULS_data_type` (see §2.5.4). It is used to hold the factors and other data about the problem being solved. It must have been initialized by a call to `ULS_initialize` and not altered by the user in the interim.

`control` is scalar `INTENT(IN)` argument of type `ULS_control_type`. Its components control the action of the factorization phase, as explained in §2.5.2.

`inform` is a scalar `INTENT(INOUT)` argument of type `ULS_inform_type` (see §2.5.3). A successful call is indicated when the component `status` has the value 0. For other return values of `status`, see §2.7.

### 2.6.3 The solution subroutine

Given the factorization, a set of equations may be solved as follows:

```
CALL ULS_solve( matrix, RHS, X, data, control, inform, trans )
```

`matrix` is scalar `INTENT(IN)` argument of type `SMT_type` that is used to specify $\mathbf{A}$. The user must set all of the relevant components of `matrix` according to the storage scheme desired (see §2.5.1). Those components set for `ULS_factorize` must not have been altered in the interim.

`RHS` is an `INTENT(IN)` assumed-shape array argument of rank 1 and of type `REAL(rp_)`. On entry, `RHS` must be set to the vector $\mathbf{b}$ and on successful return it holds the solution $\mathbf{x}$. The `i`-th component of $\mathbf{b}$ occupies the `i`-th component of `RHS`.

`X` is an `INTENT(INOUT)` assumed-shape array argument of rank 1 and of type `REAL(rp_)`. On successful return it holds the solution $\mathbf{x}$. The `i`-th component of the solution $\mathbf{x}$ occupies the `i`-th component of `X`.

`data` is a scalar `INTENT(INOUT)` argument of type `ULS_data_type` (see §2.5.4). It is used to hold the factors and other data about the problem being solved. It must have been initialized by a call to `ULS_initialize` and not altered by the user in the interim.

`control` is scalar `INTENT(IN)` argument of type `ULS_control_type`. Its components control the action of the solve phase, as explained in §2.5.2.

`inform` is a scalar `INTENT(INOUT)` argument of type `ULS_inform_type` (see §2.5.3). A successful call is indicated when the component `status` has the value 0. For other return values of `status`, see §2.7.

`trans` is a scalar `INTENT(IN)` argument of type `LOGICAL` that should be set `.TRUE.` if the solution to $\mathbf{A}^T\mathbf{x} = \mathbf{b}$ is sought, and `.FALSE.` if the solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ is required.

### 2.6.4 The termination subroutine

All previously allocated internal arrays are deallocated and OpenMP locks destroyed as follows:

```
CALL ULS_terminate( data, control, inform )
```

`data` is a scalar `INTENT(INOUT)` argument of type `ULS_data_type` (see §2.5.4). It is used to hold the factors and other data about the problem being solved. It must have been initialized by a call to `ULS_initialize` and not altered by the user in the interim. On exit, its allocatable array components will have been deallocated.

`control` is scalar `INTENT(IN)` argument of type `ULS_control_type`. Its components control the action of the termination phase, as explained in §2.5.2.

`inform` is a scalar `INTENT(INOUT)` argument of type `ULS_inform_type` (see §2.5.3). A successful call is indicated when the component `status` has the value 0. For other return values of `status`, see §2.7.

## 2.7 Warning and error messages

A negative value of `inform%status` on exit from the subroutines indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

`-1` An allocation error occurred. A message indicating the offending array is written on unit `control%error`, and the returned allocation status and a string containing the name of the offending array are held in `inform%alloc_status` and `inform%bad_alloc` respectively.

-2 A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc respectively.

-3 One of the restrictions matrix%m $>0$ or matrix%n $>0$ or matrix%ne $<0$, for co-ordinate entry, or requirements that matrix%type contain its relevant string 'COORDINATE', 'SPARSE_BY_ROWS' or 'DENSE' has been violated.

-26 The requested solver is not available.

-29 This option is not available with this solver.

-32 The integer workspace required is larger than max_factor_size.

-33 The real workspace required is larger than max_factor_size.

-39 The input permutation/pivot order is not a permutation or is faulty in some other way.

-50 A solver-specific error occurred; check the solver-specific information component of inform along with the solver's documentation for more details.

### 2.8 Further features

In this section, we describe a feature for enquiring about the factorization constructed. This feature will not be needed by a user who wants simply to solve systems of equations with matrix $\mathbf{A}$ or $\mathbf{A}^T$.

The solvers used each produce an $\mathbf{P}_R\mathbf{L}\mathbf{U}\mathbf{P}_C$ factorization of $\mathbf{A}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, and $\mathbf{P}_R$ and $\mathbf{P}_C$ are row and column permutation matrices respectively. The following subroutine is provided:

ULS_enquire returns the row and column permutations that define $\mathbf{P}_R$ and $\mathbf{P}_C$.

Support for this feature from the solvers available with ULS is summarised in Table 2.2.

| solver | ULS_enquire |
|---|---|
| GLS/MA28 | √ |
| HSL_MA48 | √ |
| GETR | √ |

Table 2.2: Options supported.

### 2.8.1 To return $P_R$ and $P_C$

```
CALL ULS_enquire( data, inform, ROWS, COLS )
```

`data` is a scalar `INTENT(INOUT)` argument of type `ULS_data_type` (see §2.5.4). It is used to hold the factors and other data about the problem being solved. It must have been initialized by a call to `ULS_initialize` and not altered by the user in the interim.

`inform` is a scalar `INTENT(INOUT)` argument of type `ULS_inform_type` (see §2.5.3). A successful call is indicated when the component `status` has the value 0. For other return values of `status`, see §2.7.

`ROWS` is a rank-one `INTEGER(ip_)` array argument of `INTENT(OUT)` and length $m$. The `ROW(`$i$`)`th row of $\mathbf{A}$ is the $i$th row in the factors, $1 \leq i \leq m$.

`COLS` is a rank-one `INTEGER(ip_)` array argument of `INTENT(OUT)` and length $n$. The `COL(`$j$`)`th column of $\mathbf{A}$ is the $j$th column in the factors, $1 \leq j \leq n$.

## 2.9 Setting control parameters

In this section, we describe an alternative means of setting control parameters, that is components of the variable `control` of type `ULS_control_type` (see §2.5.2), by reading an appropriate data specification file using the subroutine `ULS_read_specfile`. This facility is useful as it allows a user to change `ULS` control parameters without editing and recompiling programs that call `ULS`.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specification file is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by `ULS_read_specfile` must start with a "`BEGIN ULS`" command and end with an "`END`" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by ULS_read_specfile .. )
  BEGIN ULS
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by ULS_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "`BEGIN ULS`" and "`END`" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN ULS SPECIFICATION
```

and

```
END ULS SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN ULS" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy way to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameter may be of three different types, namely integer, character or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively).

The specification file must be open for input when ULS_read_specfile is called, and the associated unit number passed to the routine in device (see below). Note that the corresponding file is rewound, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by ULS_read_specfile.

### 2.9.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL ULS_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type ULS_control_type (see §2.5.2). Default values should have already been set, perhaps by calling ULS_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see §2.5.2) of control that each affects are given in Table 2.3.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specification file has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

## 3 GENERAL INFORMATION

**Workspace:** Provided automatically by the module.

**Other modules used directly:** GALAHAD_SYMBOLS, GALAHAD_SORT_single/double, GALAHAD_SPACE_single/double, GALAHAD_SPECFILE_single/double, GALAHAD_STRING_single/double, GALAHAD_SMT_single/double, GALAHAD_GLS_single/double and optionally HSL_MA48_single/double.

**Other routines called directly:** None.

**Input/output:** Output is under control of the arguments control%error, control%warning, control%out, control%statistics and control%print_level.

**Restrictions:** matrix%n $\geq 1$, matrix%ne $\geq 0$ if matrix%type = 'COORDINATE', matrix%type one of 'COORDINATE', 'SPARSE_BY_ROWS' or 'DENSE'.

**Portability:** ISO Fortran 95 + TR 15581 or Fortran 2003 and optionally OpenNP. The package is thread-safe.

| command | component of control | value type |
|---|---|---|
| error-printout-device | %error | integer |
| warning-printout-device | %warning | integer |
| printout-device | %out | integer |
| print-level | %print_level | integer |
| print-level-solver | %print_level_solver | integer |
| maximum-block-size-for-btf | %maximum_block_size_for_btf | integer |
| blas-block-for-size-factorize | %blas_block_size_factorize | integer |
| blas-block-size-for-solve | %blas_block_size_solve | integer |
| initial-fill-in-factor | %initial_fill_in_factor | integer |
| minimum-real-factor-size | %min_real_factor_size | integer |
| minimum-integer-factor-size | %min_integer_factor_size | integer |
| maximum-factor-size | %max_factor_size | integer(long) |
| pivot-control | %pivot_control | integer |
| pivot-search-limit | %pivot_search_limit | integer |
| max-iterative-refinements | %max_iterative_refinements | integer |
| stop-if-singular | %stop_if_singular | logical |
| array-increase-factor | %array_increase_factor | real |
| array-decrease-factor | %array_decrease_factor | real |
| relative-pivot-tolerance | %relative_pivot_tolerance | real |
| absolute-pivot-tolerance | %absolute_pivot_tolerance | real |
| zero-tolerance | %zero_tolerance | real |
| switch-to-full-code-density | %switch_to_full_code_density | real |
| acceptable-residual-relative | %acceptable_residual_relative | real |
| acceptable-residual-absolute | %acceptable_residual_absolute | real |
| output-line-prefix | %prefix | character |

Table 2.3: Specfile commands and associated components of control.

## 4 METHOD

Variants of sparse Gaussian elimination are used.

The solver GLS is available as part of GALAHAD and relies on the HSL Archive packages MA33. To obtain HSL Archive packages, see

http://hsl.rl.ac.uk/archive/ .

The solver HSL_MA48 is part of HSL 2007. To obtain HSL 2007 packages, see

http://hsl.rl.ac.uk/hsl2007/ .

### References:

The methods used are described in the user-documentation for

HSL 2007, A collection of Fortran codes for large-scale scientific computation (2007).
http://www.cse.clrc.ac.uk/nag/hsl

## 5 EXAMPLE OF USE

We illustrate the use of the package on the solution of the single set of equations

$$\begin{pmatrix} 11 & 12 & \\ 21 & 22 & 23 \\ & 32 & 33 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 23 \\ 66 \\ 65 \end{pmatrix}$$

(Note that this example does not illustrate all the facilities). Then, choosing the solver GLS, we may use the following code:

```
   PROGRAM GALAHAD_ULS_example  ! GALAHAD 4.2 - 2023-12-23 AT 08:00 GMT.
   USE GALAHAD_ULS_DOUBLE
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
   INTEGER :: info
   INTEGER, PARAMETER :: m = 3
   INTEGER, PARAMETER :: n = 3
   INTEGER, PARAMETER :: ne = 7
   TYPE ( SMT_type ) :: matrix
   TYPE ( ULS_data_type ) :: data
   TYPE ( ULS_control_type ) control
   TYPE ( ULS_inform_type ) :: inform
   INTEGER :: ROWS( m ), COLS( n )
   REAL ( KIND = wp ) :: X( n ), B( m )
!  Record matrix order and number of entries
   matrix%m = m ; matrix%n = n ; matrix%ne = ne
! Allocate and set matrix
   CALL SMT_put( matrix%type, 'COORDINATE', info )   ! Specify co-ordinate
   ALLOCATE( matrix%val( ne ),  matrix%row( ne ),  matrix%col( ne ) )
   matrix%row( : ne ) = (/ 1, 2, 3, 2, 1, 3, 2 /)
   matrix%col( : ne ) = (/ 1, 3, 3, 1, 2, 2, 2 /)
   matrix%val( : ne ) = (/ 11.0_wp, 23.0_wp, 33.0_wp, 21.0_wp, 12.0_wp,        &
                           32.0_wp, 22.0_wp /)
! Specify the solver (in this case gls)
   CALL ULS_initialize( 'gls', data, control, inform )
! Factorize the matrix
```

```
   CALL ULS_factorize( matrix, data, control, inform )
   IF ( inform%status < 0 ) THEN
     WRITE( 6, '( A, I0 )' )                                              &
       ' Failure of ULS_factorize with inform%status = ',  inform%status
     STOP
   END IF
! Write row and column reorderings
   CALL ULS_enquire( data, inform, ROWS, COLS )
   WRITE( 6, "( A, /, ( 10I5 ) )" ) ' row orderings:', ROWS( : inform%rank )
   WRITE( 6, "( A, /, ( 10I5 ) )" ) ' column orderings:', COLS( : inform%rank )
! set right-hand side and solve system
   B = (/ 23.0_wp, 66.0_wp, 65.0_wp /)
   CALL ULS_solve( matrix, B, X, data, control, inform, .FALSE. )
   IF ( inform%status == 0 ) WRITE( 6, '( A, /,( 6ES11.3 ) )' )           &
     ' Solution of set of equations without refinement is', X
! Clean up
   CALL ULS_terminate( data, control, inform )
   DEALLOCATE( matrix%val, matrix%row, matrix%col )
   STOP
   END PROGRAM GALAHAD_ULS_example
```

This produces the following output:

```
 row orderings:
    1    2    3
 column orderings:
    1    3    2
 Solution of set of equations without refinement is
  1.000E+00  1.000E+00  1.000E+00
```