



C interfaces to GALAHAD SLS

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Tue May 2 2023

| | |
|--|----------|
| 1 GALAHAD C package sha | 1 |
| 1.1 Introduction | 1 |
| 1.1.1 Purpose | 1 |
| 1.1.2 Authors | 1 |
| 1.1.3 Originally released | 1 |
| 2 File Index | 3 |
| 2.1 File List | 3 |
| 3 File Documentation | 5 |
| 3.1 galahad_sha.h File Reference | 5 |
| 3.1.1 Data Structure Documentation | 5 |
| 3.1.1.1 struct sha_control_type | 5 |
| 3.1.1.2 struct sha_inform_type | 6 |

Chapter 1

GALAHAD C package sls

1.1 Introduction

1.1.1 Purpose

This package **solves dense or sparse symmetric systems of linear equations** using variants of Gaussian elimination. Given a sparse symmetric $n \times n$ matrix A , and an n -vector b , this subroutine solves the system $Ax = b$. The matrix A need not be definite.

The package provides a common interface to a variety of well-known solvers from HSL and elsewhere. Currently supported solvers include MA27/SILS, HSL_MA57, HSL_MA77, HSL_MA86, HSL_MA87 and HSL_MA97 from HSL, SSIDS from SPRAL, MUMPS from Mumps Technologies, PARDISO both from the Pardiso Project and Intel's MKL, PaStix from Inria and WSMP from the IBM alpha Works, as well as POTR, SYTR and SBTR 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 than via this package.

1.1.2 Authors

N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England.

C interface, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

Julia interface, additionally A. Montoison and D. Orban, Polytechnique Montréal.

1.1.3 Originally released

August 2009, C interface December 2021.

1.1.4 Terminology

The solvers used each produce an LDL^T factorization of A or a perturbation thereof, where L is a permuted lower triangular matrix and D is a block diagonal matrix with blocks of order 1 and 2. It is convenient to write this factorization in the form

$$A + E = PLDL^T P^T,$$

where P is a permutation matrix and E is any diagonal perturbation introduced.

1.1.5 Supported external solvers

The key features of the external solvers supported by sls are given in the following table.

Table 1.1 External solver characteristics

| solver | factorization | indefinite A | out-of-core | parallelised |
|-------------|--------------------|----------------|-------------|----------------------|
| SILS/MA27 | multifrontal | yes | no | no |
| HSL_MA57 | multifrontal | yes | no | no |
| HSL_MA77 | multifrontal | yes | yes | OpenMP core |
| HSL_MA86 | left-looking | yes | no | OpenMP fully |
| HSL_MA87 | left-looking | no | no | OpenMP fully |
| HSL_MA97 | multifrontal | yes | no | OpenMP core |
| SSIDS | multifrontal | yes | no | CUDA core |
| MUMPS | multifrontal | yes | optionally | MPI |
| PARDISO | left-right-looking | yes | no | OpenMP fully |
| MKL_PARDISO | left-right-looking | yes | optionally | OpenMP fully |
| PaStix | left-right-looking | yes | no | OpenMP fully |
| WSMP | left-right-looking | yes | no | OpenMP fully |
| POTR | dense | no | no | with parallel LAPACK |
| SYTR | dense | yes | no | with parallel LAPACK |
| PBTR | dense band | no | no | with parallel LAPACK |

1.1.6 Method

Variants of sparse Gaussian elimination are used.

The solver SILS is available as part of GALAHAD and relies on the HSL Archive package MA27. To obtain HSL Archive packages, see

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

The solvers HSL_MA57, HSL_MA77, HSL_MA86, HSL_MA87 and HSL_MA97, the ordering packages MC61 and HSL_MC68, and the scaling packages HSL_MC64 and MC77 are all part of HSL 2011. To obtain HSL 2011 packages, see

<http://hsl.rl.ac.uk>

The solver SSIDS is from the SPRAL sparse-matrix collection, and is available as part of GALAHAD.

The solver MUMPS is available from Mumps Technologies in France, and version 5.5.1 or above is sufficient. To obtain MUMPS, see

<https://mumps-solver.org>.

The solver PARDISO is available from the Pardiso Project; version 4.0.0 or above is required. To obtain PARDISO, see

<http://www.pardiso-project.org/>.

The solver MKL PARDISO is available as part of Intel's oneAPI Math Kernel Library (oneMKL). To obtain this version of PARDISO, see

<https://software.intel.com/content/www/us/en/develop/tools/oneapi.html>.

The solver PaStix is available from Inria in France, and version 6.2 or above is sufficient. To obtain PaStiX, see

<https://solverstack.gitlabpages.inria.fr/pastix>.

The solver WSMP is available from the IBM alpha Works; version 10.9 or above is required. To obtain WSMP, see <http://www.alphaworks.ibm.com/tech/wsmp>.

The solvers POTR, SYTR and PBTR, are available as S/DPOTRF/S, S/DSYTRF/S and S/DPBTRF/S as part of LAPACK. Reference versions are provided by GALAHAD, but for good performance machined-tuned versions should be used.

Explicit sparsity re-orderings are obtained by calling the HSL package HSL_MC68. Both this, HSL_MA57 and PARDISO rely optionally on the ordering package MeTiS (version 4) from the Karypis Lab. To obtain METIS, see <http://glaros.dtc.umn.edu/gkhome/views/metis/>.

Bandwidth, Profile and wavefront reduction is supported by calling HSL's MC61.

1.1.7 Reference

The methods used are described in the user-documentation for

HSL 2011, A collection of Fortran codes for large-scale scientific computation (2011). <http://www.hsl.rl.ac.uk>

and papers

E. Agullo, P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, A. Guermouche and F.-H. Rouet, "Robust memory-aware mappings for parallel multifrontal factorizations". SIAM Journal on Scientific Computing, **38**(3) (2016), C256–C279,

P. R. Amestoy, I. S. Duff, J. Koster and J.-Y. L'Excellent. "A fully asynchronous multifrontal solver using distributed dynamic scheduling". SIAM Journal on Matrix Analysis and Applications **23**(1) (2001) 15-41,

A. Gupta, "WSMP: Watson Sparse Matrix Package Part I - direct solution of symmetric sparse systems". IBM Research Report RC 21886, IBM T. J. Watson Research Center, NY 10598, USA (2010),

P. Henon, P. Ramet and J. Roman, "PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems". Parallel Computing, **28**(2) (2002) 301–321,

J.D. Hogg, E. Ovtchinnikov and J.A. Scott. "A sparse symmetric indefinite direct solver for GPU architectures". ACM Transactions on Mathematical Software **42**(1) (2014), Article 1,

O. Schenk and K. Gärtner, "Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO". Journal of Future Generation Computer Systems, **20**(3) (2004) 475–487, and

O. Schenk and K. Gärtner, "On fast factorization pivoting methods for symmetric indefinite systems". Electronic Transactions on Numerical Analysis **23** (2006) 158–179.

1.1.8 Call order

To solve a given problem, functions from the sls package must be called in the following order:

- [sls_initialize](#) - provide default control parameters and set up initial data structures
- [sls_read_specfile](#) (optional) - override control values by reading replacement values from a file
- [sls_analyse_matrix](#) - set up matrix data structures and analyse the structure to choose a suitable order for factorization
- [sls_reset_control](#) (optional) - possibly change control parameters if a sequence of problems are being solved
- [sls_factorize_matrix](#) - form and factorize the matrix A
- one of
 - [sls_solve_system](#) - solve the linear system of equations $Ax = b$
 - [sls_partial_solve_system](#) - solve a linear system $Mx = b$ involving one of the matrix factors M of A
- [sls_information](#) (optional) - recover information about the solution and solution process
- [sls_terminate](#) - deallocate data structures

See Section ?? for examples of use.

1.1.9 Symmetric matrix storage formats

The symmetric n by n coefficient matrix A may be presented and stored in a variety of convenient input formats. Crucially symmetry is exploited by only storing values from the lower triangular part (i.e, those entries that lie on or below the leading diagonal).

Both C-style (0 based) and fortran-style (1-based) indexing is allowed. Choose `control.f_indexing` as `false` for C style and `true` for fortran style; the discussion below presumes C style, but add 1 to indices for the corresponding fortran version.

Wrappers will automatically convert between 0-based (C) and 1-based (fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing.

1.1.9.1 Dense storage format

The matrix 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. Since A is symmetric, only the lower triangular part (that is the part A_{ij} for $0 \leq j \leq i \leq n-1$) need be held. In this case the lower triangle should be stored by rows, that is component $i * i/2 + j$ of the storage array `val` will hold the value A_{ij} (and, by symmetry, A_{ji}) for $0 \leq j \leq i \leq n-1$.

1.1.9.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l -th entry, $0 \leq l \leq ne-1$, of A , its row index i , column index j and value A_{ij} , $0 \leq j \leq i \leq n-1$, are stored as the l -th components of the integer arrays `row` and `col` and real array `val`, respectively, while the number of nonzeros is recorded as `ne = ne`. Note that only the entries in the lower triangle should be stored.

1.1.9.3 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 A the i -th component of the integer array `ptr` holds the position of the first entry in this row, while `ptr(n)` holds the total number of entries. The column indices j , $0 \leq j \leq i$, and values A_{ij} of the entries in the i -th row are stored in components $l = \text{ptr}(i), \dots, \text{ptr}(i+1)-1$ of the integer array `col`, and real array `val`, respectively. Note that as before only the entries in the lower triangle should be stored. For sparse matrices, this scheme almost always requires less storage than its predecessor.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | | |
|-------------------------------|-------|----|
| galahad_sls.h | | ?? |
|-------------------------------|-------|----|

Chapter 3

File Documentation

3.1 galahad_sls.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sils.h"
#include "hsl_ma57.h"
#include "hsl_ma77.h"
#include "hsl_ma86.h"
#include "hsl_ma87.h"
#include "hsl_ma97.h"
#include "spral_ssids.h"
#include "hsl_mc64.h"
#include "hsl_mc68.h"
```

Data Structures

- struct [sls_control_type](#)
- struct [sls_time_type](#)
- struct [sls_inform_type](#)

Functions

- void [sls_initialize](#) (const char solver[], void **data, struct [sls_control_type](#) *control, int *status)
- void [sls_read_specfile](#) (struct [sls_control_type](#) *control, const char specfile[])
- void [sls_analyse_matrix](#) (struct [sls_control_type](#) *control, void **data, int *status, int n, const char type[], int ne, const int row[], const int col[], const int ptr[])
- void [sls_reset_control](#) (struct [sls_control_type](#) *control, void **data, int *status)
- void [sls_factorize_matrix](#) (void **data, int *status, int ne, const real_wp_ val[])
- void [sls_solve_system](#) (void **data, int *status, int n, real_wp_ sol[])
- void [sls_partial_solve_system](#) (const char part[], void **data, int *status, int n, real_wp_ sol[])
- void [sls_information](#) (void **data, struct [sls_inform_type](#) *inform, int *status)
- void [sls_terminate](#) (void **data, struct [sls_control_type](#) *control, struct [sls_inform_type](#) *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct sls_control_type

control derived type as a C struct

Examples

[slst.c](#), and [slstf.c](#).

Data Fields

| | | |
|----------|---------------------------|--|
| bool | f_indexing | use C or Fortran sparse matrix indexing |
| int | error | unit for error messages |
| int | warning | unit for warning messages |
| int | out | unit for monitor output |
| int | statistics | unit for statistical output |
| int | print_level | controls level of diagnostic output |
| int | print_level_solver | controls level of diagnostic output from external solver |
| int | bits | number of bits used in architecture |
| int | block_size_kernel | the target blocksize for kernel factorization |
| int | block_size_elimination | the target blocksize for parallel elimination |
| int | blas_block_size_factorize | level 3 blocking in factorize |
| int | blas_block_size_solve | level 2 and 3 blocking in solve |
| int | node_amalgamation | a child node is merged with its parent if they both involve fewer than node_amalgamation eliminations |
| int | initial_pool_size | initial size of task-pool arrays for parallel elimination |
| int | min_real_factor_size | initial size for real array for the factors and other data |
| int | min_integer_factor_size | initial size for integer array for the factors and other data |
| int64_t | max_real_factor_size | maximum size for real array for the factors and other data |
| int64_t | max_integer_factor_size | maximum size for integer array for the factors and other data |
| int64_t | max_in_core_store | amount of in-core storage to be used for out-of-core factorization |
| real_wp_ | array_increase_factor | factor by which arrays sizes are to be increased if they are too small |
| real_wp_ | array_decrease_factor | if previously allocated internal workspace arrays are greater than array_decrease_factor times the currently required sizes, they are reset to current requirements |
| int | pivot_control | pivot control: <ul style="list-style-type: none"> • 1 Numerical pivoting will be performed. • 2 No pivoting will be performed and an error exit will occur immediately a pivot sign change is detected. • 3 No pivoting will be performed and an error exit will occur if a zero pivot is detected. • 4 No pivoting is performed but pivots are changed to all be positive |

Data Fields

| | | |
|----------|--------------------------|---|
| int | ordering | controls ordering (ignored if explicit PERM argument present) <ul style="list-style-type: none"> • <0 chosen by the specified solver with its own ordering-selected value -ordering • 0 chosen package default (or the AMD ordering if no package default) • 1 Approximate minimum degree (AMD) with provisions for "dense" rows/col • 2 Minimum degree • 3 Nested dissection • 4 indefinite ordering to generate a combination of 1x1 and 2x2 pivots • 5 Profile/Wavefront reduction • 6 Bandwidth reduction • >6 ordering chosen depending on matrix characteristics (not yet implemented) |
| int | full_row_threshold | controls threshold for detecting full rows in analyse, registered as percentage of matrix order. If 100, only fully dense rows detected (defa |
| int | row_search_indefinite | number of rows searched for pivot when using indefinite ordering |
| int | scaling | controls scaling (ignored if explicit SCALE argument present) <ul style="list-style-type: none"> • <0 chosen by the specified solver with its own scaling-selected value -scaling • 0 No scaling • 1 Scaling using HSL's MC64 • 2 Scaling using HSL's MC77 based on the row one-norm • 3 Scaling using HSL's MC77 based on the row infinity-norm |
| int | scale_maxit | the number of scaling iterations performed (default 10 used if .scale_maxit < 0) |
| real_wp_ | scale_thresh | the scaling iteration stops as soon as the row/column norms are less than 1+/-scale_thresh |
| real_wp_ | relative_pivot_tolerance | pivot threshold |
| real_wp_ | minimum_pivot_tolerance | smallest permitted relative pivot threshold |
| real_wp_ | absolute_pivot_tolerance | any pivot small than this is considered zero |
| real_wp_ | zero_tolerance | any entry smaller than this is considered zero |
| real_wp_ | zero_pivot_tolerance | any pivot smaller than this is considered zero for positive-definite sol |
| real_wp_ | negative_pivot_tolerance | any pivot smaller than this is considered to be negative for p-d solvers |

Data Fields

| | | |
|----------|--------------------------------------|---|
| real_wp_ | static_pivot_tolerance | used for setting static pivot level |
| real_wp_ | static_level_switch | used for switch to static |
| real_wp_ | consistency_tolerance | used to determine whether a system is consistent when seeking a Fredholm alternative |
| int | max_iterative_refinements | maximum number of iterative refinements allowed |
| real_wp_ | acceptable_residual_relative | refinement will cease as soon as the residual $\ Ax-b\ $ falls below $\max(\text{acceptable_residual_relative} * \ b\ , \text{acceptable_residual_absolute})$ |
| real_wp_ | acceptable_residual_absolute | see acceptable_residual_relative |
| bool | multiple_rhs | set .multiple_rhs to .true. if there is possibility that the solver will be required to solve systems with more than one right-hand side. More efficient execution may be possible when .multiple_rhs = .false. |
| bool | generate_matrix_file | if .generate_matrix_file is .true. if a file describing the current matrix is to be generated |
| int | matrix_file_device | specifies the unit number to write the input matrix (in co-ordinate form) |
| char | matrix_file_name[31] | name of generated matrix file containing input problem |
| char | out_of_core_directory[401] | directory name for out of core factorization and additional real workspace in the indefinite case, respectively |
| char | out_of_core_integer_factor_file[401] | out of core superfile names for integer and real factor data, real works and additional real workspace in the indefinite case, respectively |
| char | out_of_core_real_factor_file[401] | see out_of_core_integer_factor_file |
| char | out_of_core_real_work_file[401] | see out_of_core_integer_factor_file |
| char | out_of_core_indefinite_file[401] | see out_of_core_integer_factor_file |
| char | out_of_core_restart_file[501] | see out_of_core_integer_factor_file |
| char | prefix[31] | all output lines will be prefixed by prefix(2:LEN(TRIM(.prefix))-1) where prefix contains the required string enclosed in quotes, e.g. "string" or 'string' |

3.1.1.2 struct sls_time_type

time derived type as a C struct

Data Fields

| | | |
|----------|--------------------|--|
| real_wp_ | total | the total cpu time spent in the package |
| real_wp_ | analyse | the total cpu time spent in the analysis phase |
| real_wp_ | factorize | the total cpu time spent in the factorization phase |
| real_wp_ | solve | the total cpu time spent in the solve phases |
| real_wp_ | order_external | the total cpu time spent by the external solver in the ordering phase |
| real_wp_ | analyse_external | the total cpu time spent by the external solver in the analysis phase |
| real_wp_ | factorize_external | the total cpu time spent by the external solver in the factorization phase |
| real_wp_ | solve_external | the total cpu time spent by the external solver in the solve phases |
| real_wp_ | clock_total | the total clock time spent in the package |
| real_wp_ | clock_analyse | the total clock time spent in the analysis phase |
| real_wp_ | clock_factorize | the total clock time spent in the factorization phase |

Data Fields

| | | |
|----------|--------------------------|--|
| real_wp_ | clock_solve | the total clock time spent in the solve phases |
| real_wp_ | clock_order_external | the total clock time spent by the external solver in the ordering phase |
| real_wp_ | clock_analyse_external | the total clock time spent by the external solver in the analysis phase |
| real_wp_ | clock_factorize_external | the total clock time spent by the external solver in the factorization p |
| real_wp_ | clock_solve_external | the total clock time spent by the external solver in the solve phases |

3.1.1.3 struct sls_inform_type

inform derived type as a C struct

Examples

[slst.c](#), and [slstf.c](#).

Data Fields

| | | | |
|--|---------|-------------------------|--|
| | int | status | reported return status: 0 success -1 allocation error -2 deallocation error -3 matrix data faulty (.n < 1, .ne < 0) -20 allegedly +ve definite matrix is not -29 unavailable option -31 input order is not a permutation or is faulty in some other way -32 > control.max_integer_factor_size integer space required for factor -33 > control.max_real_factor_size real space required for factors -40 not possible to alter the diagonals -41 no access to permutation or pivot sequence used -42 no access to diagonal perturbations -43 direct-access file error -50 solver-specific error; see the solver's info parameter -101 unknown solver |
| | int | alloc_status | STAT value after allocate failure. |
| | char | bad_alloc[81] | name of array which provoked an allocate failure |
| | int | more_info | further information on failure |
| | int | entries | number of entries |
| | int | out_of_range | number of indices out-of-range |
| | int | duplicates | number of duplicates |
| | int | upper | number of entries from the strict upper triangle |
| | int | missing_diagonals | number of missing diagonal entries for an allegedly-definite matrix |
| | int | max_depth_assembly_tree | maximum depth of the assembly tree |
| | int | nodes_assembly_tree | nodes in the assembly tree (= number of elimination steps) |
| | int64_t | real_size_desirable | desirable or actual size for real array for the factors and other data |
| | int64_t | integer_size_desirable | desirable or actual size for integer array for the factors and other dat |
| | int64_t | real_size_necessary | necessary size for real array for the factors and other data |
| | int64_t | integer_size_necessary | necessary size for integer array for the factors and other data |
| | int64_t | real_size_factors | predicted or actual number of reals to hold factors |

Data Fields

| | | |
|------------------------|------------------------|--|
| int64_t | integer_size_factors | predicted or actual number of integers to hold factors |
| int64_t | entries_in_factors | number of entries in factors |
| int | max_task_pool_size | maximum number of tasks in the factorization task pool |
| int | max_front_size | forecast or actual size of largest front |
| int | compresses_real | number of compresses of real data |
| int | compresses_integer | number of compresses of integer data |
| int | two_by_two_pivots | number of 2x2 pivots |
| int | semi_bandwidth | semi-bandwidth of matrix following bandwidth reduction |
| int | delayed_pivots | number of delayed pivots (total) |
| int | pivot_sign_changes | number of pivot sign changes if no pivoting is used successfully |
| int | static_pivots | number of static pivots chosen |
| int | first_modified_pivot | first pivot modification when static pivoting |
| int | rank | estimated rank of the matrix |
| int | negative_eigenvalues | number of negative eigenvalues |
| int | num_zero | number of pivots that are considered zero (and ignored) |
| int | iterative_refinements | number of iterative refinements performed |
| int64_t | flops_assembly | anticipated or actual number of floating-point operations in assembly |
| int64_t | flops_elimination | anticipated or actual number of floating-point operations in elimination |
| int64_t | flops_blas | additional number of floating-point operations for BLAS |
| real_wp_ | largest_modified_pivot | largest diagonal modification when static pivoting or ensuring definiten |
| real_wp_ | minimum_scaling_factor | minimum scaling factor |
| real_wp_ | maximum_scaling_factor | maximum scaling factor |
| real_wp_ | condition_number_1 | esimate of the condition number of the matrix (category 1 equations) |
| real_wp_ | condition_number_2 | estimate of the condition number of the matrix (category 2 equations) |
| real_wp_ | backward_error_1 | esimate of the backward error (category 1 equations) |
| real_wp_ | backward_error_2 | esimate of the backward error (category 2 equations) |
| real_wp_ | forward_error | estimate of forward error |
| bool | alternative | has an "alternative" y : $A y = 0$ and $y^T b > 0$ been found when trying to solve $A x = b$? |
| char | solver[21] | name of external solver used to factorize and solve |
| struct sils_time_type | time | timings (see above) |
| struct sils_ainfo_type | sils_ainfo | the output structure from sils |
| struct sils_finfo_type | sils_finfo | see sils_ainfo |
| struct sils_sinfo_type | sils_sinfo | see sils_ainfo |
| struct ma57_ainfo | ma57_ainfo | the output structure from ma57 |
| struct ma57_finfo | ma57_finfo | see ma57_ainfo |

Data Fields

| | | |
|---------------------------|-----------------------|--|
| struct ma57_sinfo | ma57_sinfo | see ma57_ainfo |
| struct ma77_info | ma77_info | the output structure from ma77 |
| struct ma86_info | ma86_info | the output structure from ma86 |
| struct ma87_info | ma87_info | the output structure from ma87 |
| struct ma97_info | ma97_info | the output structure from ma97 |
| struct spral_ssids_inform | ssids_inform | the output structure from ssids |
| int | mc61_info[10] | the integer and real output arrays from mc61 |
| real_wp_ | mc61_rinfo[15] | see mc61_info |
| struct mc64_info | mc64_info | the output structure from mc64 |
| struct mc68_info | mc68_info | the output structure from mc68 |
| int | mc77_info[10] | the integer output array from mc77 |
| real_wp_ | mc77_rinfo[10] | the real output status from mc77 |
| int | mumps_error | the output scalars and arrays from mumps |
| int | mumps_info[80] | see mumps_error |
| real_wp_ | mumps_rinfo[40] | see mumps_error |
| int | pardiso_error | the output scalars and arrays from pardiso |
| int | pardiso_IPARM[64] | see pardiso_error |
| real_wp_ | pardiso_DPARM[64] | see pardiso_error |
| int | mkl_pardiso_error | the output scalars and arrays from mkl_pardiso |
| int | mkl_pardiso_IPARM[64] | see mkl_pardiso_error |
| int | pastix_info | the output flag from pastix |
| int | wsmp_error | the output scalars and arrays from wsmp |
| int | wsmp_iparm[64] | see wsmp_error |
| real_wp_ | wsmp_dparm[64] | see wsmp_error |
| int | mpi_ierr | the output flag from MPI routines |
| int | lapack_error | the output flag from LAPACK routines |

3.1.2 Function Documentation

3.1.2.1 sls_initialize()

```
void sls_initialize (
    const char solver[],
    void ** data,
    struct sls_control_type * control,
    int * status )
```

Select solver, set default control values and initialize private data

Parameters

| | | |
|---------|----------------|---|
| in | <i>solver</i> | is a one-dimensional array of type char that specifies the solver package that should be used to factorize the matrix A . It should be one of 'sils', 'ma27', 'ma57', 'ma77', 'ma86', 'ma87', 'ma97', 'ssids', 'mumps', 'pardiso', 'mkl pardiso', 'pastix', 'wsmp', 'potr', 'sytr' or 'pbtr'; lower or upper case variants are allowed. |
| in, out | <i>data</i> | holds private internal data |
| out | <i>control</i> | is a struct containing control information (see sls_control_type) |
| out | <i>status</i> | is a scalar variable of type int, that gives the exit status from the package. Possible values are: <ul style="list-style-type: none"> • 0. The import was succesful. • -26. The requested solver is not available. |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.2 sls_read_specfile()

```
void sls_read_specfile (
    struct sls\_control\_type * control,
    const char specfile[] )
```

Read the content of a specification file, and assign values associated with given keywords to the corresponding control parameters. By default, the spcification file will be named RUNSLS.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in \$GALAHAD/doc/sls.pdf for a list of keywords that may be set.

Parameters

| | | |
|---------|-----------------|--|
| in, out | <i>control</i> | is a struct containing control information (see sls_control_type) |
| in | <i>specfile</i> | is a character string containing the name of the specification file |

3.1.2.3 sls_analyse_matrix()

```
void sls_analyse_matrix (
    struct sls\_control\_type * control,
    void ** data,
    int * status,
    int n,
    const char type[],
    int ne,
    const int row[],
```

```
const int col[],  
const int ptr[] )
```

Import structural matrix data into internal storage prior to solution

Parameters

| | | |
|---------|----------------|---|
| in | <i>control</i> | is a struct whose members provide control paramters for the remaining pcedures (see sls_control_type) |
| in, out | <i>data</i> | holds private internal data |
| out | <i>status</i> | <p>is a scalar variable of type int, that gives the exit status from the package. Possible values are:</p> <ul style="list-style-type: none"> • 0. The import and analysis were conducted succesfully. • -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. The restrictions $n > 0$ or requirement that the matrix type must contain the relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated. • -20. The matrix is not positive definite while the solver used expected it to be. • -26. The requested solver is not available. • -29. This option is not available with this solver. • -32. More than control.max integer factor size words of internal integer storage are required for in-core factorization. • -34. The package PARDISO failed; check the solver-specific information components inform.pardiso iparm and inform.pardiso_dparm along with PARDISO's documentation for more details. • -35. The package WSMP failed; check the solver-specific information components inform.wsmpparm and inform.wsmpparm dparm along with WSMP's documentation for more details. • -36. The scaling package HSL MC64 failed; check the solver-specific information component inform.mc64_info along with HSL MC64's documentation for more details. • -37. The scaling package MC77 failed; check the solver-specific information components inform.mc77 info and inform.mc77_rinfo along with MC77's documentation for more details. • -43. A direct-access file error occurred. See the value of inform.ma77_info.flag for more details. • -50. A solver-specific error occurred; check the solver-specific information component of inform along with the solver's documentation for more details. |
| in | <i>n</i> | is a scalar variable of type int, that holds the number of rows in the symmetric matrix A . |
| in | <i>type</i> | is a one-dimensional array of type char that specifies the symmetric storage scheme used for the matrix A . It should be one of 'coordinate', 'sparse_by_rows' or 'dense'; lower or upper case variants are allowed. |
| in | <i>ne</i> | is a scalar variable of type int, that holds the number of entries in the lower triangular part of A in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes. |

Parameters

| | | |
|----|------------|--|
| in | <i>row</i> | is a one-dimensional array of size <i>ne</i> and type <code>int</code> , that holds the row indices of the lower triangular part of A in the sparse co-ordinate storage scheme. It need not be set for any of the other three schemes, and in this case can be <code>NULL</code> . |
| in | <i>col</i> | is a one-dimensional array of size <i>ne</i> and type <code>int</code> , that holds the column indices of the lower triangular part of A in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense storage scheme is used, and in this case can be <code>NULL</code> . |
| in | <i>ptr</i> | is a one-dimensional array of size <i>n</i> +1 and type <code>int</code> , that holds the starting position of each row of the lower triangular part of A , as well as the total number of entries, in the sparse row-wise storage scheme. It need not be set when the other schemes are used, and in this case can be <code>NULL</code> . |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.4 sls_reset_control()

```
void sls_reset_control (
    struct sls_control_type * control,
    void ** data,
    int * status )
```

Reset control parameters after import if required.

Parameters

| | | |
|---------|----------------|--|
| in | <i>control</i> | is a struct whose members provide control paramters for the remaining prcedures (see sls_control_type) |
| in, out | <i>data</i> | holds private internal data |
| in, out | <i>status</i> | is a scalar variable of type <code>int</code> , that gives the exit status from the package. Possible values are: <ul style="list-style-type: none"> • 0. The import was succesful. |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.5 sls_factorize_matrix()

```
void sls_factorize_matrix (
    void ** data,
```

```

int * status,
int ne,
const real_wp_ val[] )

```

Form and factorize the symmetric matrix A .

Parameters

| in, out | data | holds private internal data |
|---------|---------------|---|
| out | <i>status</i> | <p>is a scalar variable of type int, that gives the exit status from the package. Possible values are:</p> <ul style="list-style-type: none"> • 0. The factors were generated succesfully. • -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. The restrictions $n > 0$ or requirement that the matrix type must contain the relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated. • -20. The matrix is not positive definite while the solver used expected it to be. • -26. The requested solver is not available. • -29. This option is not available with this solver. • -32. More than control.max integer factor size words of internal integer storage are required for in-core factorization. • -34. The package PARDISO failed; check the solver-specific information components inform.pardiso iparm and inform.pardiso_dparm along with PARDISO's documentation for more details. • -35. The package WSMP failed; check the solver-specific information components inform.wsmp_iparm and inform.wsmp_dparm along with WSMP's documentation for more details. • -36. The scaling package HSL MC64 failed; check the solver-specific information component inform.mc64_info along with HSL MC64's documentation for more details. • -37. The scaling package MC77 failed; check the solver-specific information components inform.mc77_info and inform.mc77_rinfo along with MC77's documentation for more details. • -43. A direct-access file error occurred. See the value of inform.ma77_info.flag for more details. • -50. A solver-specific error occurred; check the solver-specific information component of inform along with the solver's documentation for more details. |
| in | <i>ne</i> | is a scalar variable of type int, that holds the number of entries in the lower triangular part of the symmetric matrix A . |
| in | <i>val</i> | is a one-dimensional array of size ne and type double, that holds the values of the entries of the lower triangular part of the symmetric matrix A in any of the supported storage schemes. |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.6 sls_solve_system()

```
void sls_solve_system (
    void ** data,
    int * status,
    int n,
    real_wp_ sol[] )
```

Solve the linear system $Ax = b$.

Parameters

| | | |
|---------|---------------|---|
| in, out | <i>data</i> | holds private internal data |
| in, out | <i>status</i> | <p>is a scalar variable of type int, that gives the exit status from the package. Possible values are:</p> <ul style="list-style-type: none"> • 0. The required solution was obtained. • -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. • -34. The package PARDISO failed; check the solver-specific information components inform.pardiso iparm and inform.pardiso_dparm along with PARDISO's documentation for more details. • -35. The package WSMP failed; check the solver-specific information components inform.wsmp_iparm and inform.wsmp dparm along with WSMP's documentation for more details. |
| in | <i>n</i> | is a scalar variable of type int, that holds the number of entries in the vectors b and x . |
| in, out | <i>sol</i> | is a one-dimensional array of size n and type double. On entry, it must hold the vector b . On a successful exit, it contains the solution x . |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.7 sls_partial_solve_system()

```
void sls_partial_solve_system (
    const char part[],
```

```

void ** data,
int * status,
int n,
real_wp_ sol[] )

```

Given the factorization $A = LDU$ with $U = L^T$, solve the linear system $Mx = b$, where M is one of L , D , U or $S = L\sqrt{D}$.

Parameters

| | | |
|---------|---------------|--|
| in | <i>part</i> | is a one-dimensional array of type char that specifies the component M of the factorization that is to be used. It should be one of "L", "D", "U" or "S", and these correspond to the parts L , D , U and S ; lower or upper case variants are allowed. |
| in, out | <i>data</i> | holds private internal data |
| in, out | <i>status</i> | <p>is a scalar variable of type int, that gives the entry and exit status from the package. On initial entry, status must be set to 1. Possible exit are:</p> <ul style="list-style-type: none"> • 0. The required solution was obtained. • -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. • -34. The package PARDISO failed; check the solver-specific information components inform.pardiso iparm and inform.pardiso_dparm along with PARDISO's documentation for more details. • -35. The package WSMP failed; check the solver-specific information components inform.wsmpp iparm and inform.wsmpp dparm along with WSMP's documentation for more details. |
| in | <i>n</i> | is a scalar variable of type int, that holds the number of entries in the vectors b and x . |
| in, out | <i>sol</i> | is a one-dimensional array of size n and type double. On entry, it must hold the vector b . On a successful exit, it contains the solution x . |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.8 sls_information()

```

void sls_information (
    void ** data,
    struct sls_inform_type * inform,
    int * status )

```

Provide output information

Parameters

| | | |
|---------|---------------|---|
| in, out | <i>data</i> | holds private internal data |
| out | <i>inform</i> | is a struct containing output information (see sls_inform_type) |
| out | <i>status</i> | is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): <ul style="list-style-type: none"> • 0. The values were recorded succesfully |

Examples

[slst.c](#), and [slstf.c](#).

3.1.2.9 sls_terminate()

```
void sls_terminate (
    void ** data,
    struct sls_control_type * control,
    struct sls_inform_type * inform )
```

Deallocate all internal private storage

Parameters

| | | |
|---------|----------------|--|
| in, out | <i>data</i> | holds private internal data |
| out | <i>control</i> | is a struct containing control information (see sls_control_type) |
| out | <i>inform</i> | is a struct containing output information (see sls_inform_type) |

Examples

[slst.c](#), and [slstf.c](#).

Chapter 4

Example Documentation

4.1 slst.c

This is an example of how to use the package in conjunction with the sparse linear solver `sils`. A variety of supported matrix storage formats are illustrated.

Notice that C-style indexing is used, and that this is flagged by setting `control.f_indexing` to `false`.

```
/* slst.c */
/* Full test for the SLS C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sls.h"
int maxabsarray(real_wp_ a[], int n, real_wp_ *maxabs);
int main(void) {
    // Derived types
    void *data;
    struct sls_control_type control;
    struct sls_inform_type inform;
    // Set problem data
    int n = 5; // dimension of A
    int ne = 7; // number of entries of A
    int dense_ne = 15; // number of elements of A as a dense matrix
    int row[] = {0, 1, 1, 2, 2, 3, 4}; // row indices, NB lower triangle
    int col[] = {0, 0, 4, 1, 2, 2, 4}; // column indices
    int ptr[] = {0, 1, 3, 5, 6, 7}; // pointers to indices
    real_wp_val[] = {2.0, 3.0, 6.0, 4.0, 1.0, 5.0, 1.0}; // values
    real_wp_dense[] = {2.0, 3.0, 0.0, 0.0, 4.0, 1.0, 0.0,
                      0.0, 5.0, 0.0, 0.0, 6.0, 0.0, 0.0, 1.0};
    real_wp_rhs[] = {8.0, 45.0, 31.0, 15.0, 17.0};
    real_wp_sol[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int i, status;
    real_wp_x[n];
    real_wp_error[n];
    real_wp_norm_residual;
    real_wp_good_x = pow( DBL_EPSILON, 0.3333 );
    printf(" C sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    printf(" storage      RHS      refine partial\n\n");
    for( int d=1; d <= 3; d++){
        // Initialize SLS - use the sytr solver
        sls_initialize( "sytr", &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = false; // C sparse matrix indexing
        switch(d){ // import matrix data and factorize
            case 1: // sparse co-ordinate storage
                printf(" coordinate\n");
                sls_analyse_matrix( &control, &data, &status, n,
                                   "coordinate", ne, row, col, NULL );
                sls_factorize_matrix( &data, &status, ne, val );
                break;
```

```

    case 2: // sparse by rows
        printf(" sparse by rows ");
        sls_analyse_matrix( &control, &data, &status, n,
                           "sparse_by_rows", ne, NULL, col, ptr );
        sls_factorize_matrix( &data, &status, ne, val );
        break;
    case 3: // dense
        printf(" dense ");
        sls_analyse_matrix( &control, &data, &status, n,
                           "dense", ne, NULL, NULL, NULL );
        sls_factorize_matrix( &data, &status, dense_ne, dense );
        break;
}

// Set right-hand side and solve the system
for(i=0; i<n; i++) x[i] = rhs[i];
sls_solve_system( &data, &status, n, x );
sls_information( &data, &inform, &status );
if(inform.status == 0){
    for(i=0; i<n; i++) error[i] = x[i]-sol[i];
    status = maxabsarray( error, n, &norm_residual );
    if(norm_residual < good_x){
        printf(" ok ");
    }else{
        printf(" fail ");
    }
}
}
else{
    printf(" SLS_solve exit status = %li\n", inform.status);
}
//printf("sol: ");
//for( int i = 0; i < n; i++) printf("%f ", x[i]);
// resolve, this time using iterative refinement
control.max_iterative_refinements = 1;
sls_reset_control( &control, &data, &status );
for(i=0; i<n; i++) x[i] = rhs[i];
sls_solve_system( &data, &status, n, x );
sls_information( &data, &inform, &status );
if(inform.status == 0){
    for(i=0; i<n; i++) error[i] = x[i]-sol[i];
    status = maxabsarray( error, n, &norm_residual );
    if(norm_residual < good_x){
        printf(" ok ");
    }else{
        printf(" fail ");
    }
}
}
else{
    printf(" SLS_solve exit status = %li\n", inform.status);
}
}
// obtain the solution by part solves
for(i=0; i<n; i++) x[i] = rhs[i];
sls_partial_solve_system( "L", &data, &status, n, x );
sls_partial_solve_system( "D", &data, &status, n, x );
sls_partial_solve_system( "U", &data, &status, n, x );
sls_information( &data, &inform, &status );
if(inform.status == 0){
    for(i=0; i<n; i++) error[i] = x[i]-sol[i];
    status = maxabsarray( error, n, &norm_residual );
    if(norm_residual < good_x){
        printf(" ok ");
    }else{
        printf(" fail ");
    }
}
}
else{
    printf(" SLS_solve exit status = %li\n", inform.status);
}
}
// Delete internal workspace
sls_terminate( &data, &control, &inform );
printf("\n");
}
}

int maxabsarray(real_wp_ a[], int n, real_wp_ *maxabs)
{
    int i;
    real_wp_ b, max;
    max=abs(a[0]);
    for(i=1; i<n; i++)
    {
        b = fabs(a[i]);
        if(max<b)
            max=b;
    }
    *maxabs=max;
}

```

4.2 slstf.c

This is the same example, but now fortran-style indexing is used.

```

/* slstf.c */
/* Full test for the SLS C interface using Fortran sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sls.h"
int maxabsarray(real_wp_ a[], int n, real_wp_ *maxabs);
int main(void) {
    // Derived types
    void *data;
    struct sls_control_type control;
    struct sls_inform_type inform;
    // Set problem data
    int n = 5; // dimension of A
    int ne = 7; // number of entries of A
    int dense_ne = 15; // number of elements of A as a dense matrix
    int row[] = {1, 2, 2, 3, 3, 4, 5}; // row indices, NB lower triangle
    int col[] = {1, 1, 5, 2, 3, 3, 5}; // column indices
    int ptr[] = {1, 2, 4, 6, 7, 8}; // pointers to indices
    real_wp_ val[] = {2.0, 3.0, 6.0, 4.0, 1.0, 5.0, 1.0}; // values
    real_wp_ dense[] = {2.0, 3.0, 0.0, 0.0, 4.0, 1.0, 0.0,
                       0.0, 5.0, 0.0, 0.0, 6.0, 0.0, 0.0, 1.0};
    real_wp_ rhs[] = {8.0, 45.0, 31.0, 15.0, 17.0};
    real_wp_ sol[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int i, status;
    real_wp_ x[n];
    real_wp_ error[n];
    real_wp_ norm_residual;
    real_wp_ good_x = pow( DBL_EPSILON, 0.3333 );
    printf(" Fortran sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    printf(" storage      RHS      refine partial\n");
    for( int d=1; d <= 3; d++){
        // Initialize SLS - use the sytr solver
        sls_initialize( "sytr", &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = true; // Fortran sparse matrix indexing
        switch(d){ // import matrix data and factorize
            case 1: // sparse co-ordinate storage
                printf(" coordinate      ");
                sls_analyse_matrix( &control, &data, &status, n,
                                   "coordinate", ne, row, col, NULL );
                sls_factorize_matrix( &data, &status, ne, val );
                break;
            case 2: // sparse by rows
                printf(" sparse by rows ");
                sls_analyse_matrix( &control, &data, &status, n,
                                   "sparse_by_rows", ne, NULL, col, ptr );
                sls_factorize_matrix( &data, &status, ne, val );
                break;
            case 3: // dense
                printf(" dense          ");
                sls_analyse_matrix( &control, &data, &status, n,
                                   "dense", ne, NULL, NULL, NULL );
                sls_factorize_matrix( &data, &status, dense_ne, dense );
                break;
        }
        // Set right-hand side and solve the system
        for(i=0; i<n; i++) x[i] = rhs[i];
        sls_solve_system( &data, &status, n, x );
        sls_information( &data, &inform, &status );
        if(inform.status == 0){
            for(i=0; i<n; i++) error[i] = x[i]-sol[i];
            status = maxabsarray( error, n, &norm_residual );
            if(norm_residual < good_x){
                printf(" ok ");
            }else{
                printf(" fail ");
            }
        }else{
            printf(" SLS_solve exit status = %li\n", inform.status);
        }
        //printf("sol: ");
        //for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // resolve, this time using iterative refinement
        control.max_iterative_refinements = 1;
        sls_reset_control( &control, &data, &status );
        for(i=0; i<n; i++) x[i] = rhs[i];
    }
}

```

```

    sls_solve_system( &data, &status, n, x );
    sls_information( &data, &inform, &status );
    if(inform.status == 0){
        for(i=0; i<n; i++) error[i] = x[i]-sol[i];
        status = maxabsarray( error, n, &norm_residual );
        if(norm_residual < good_x){
            printf("    ok ");
        }else{
            printf("    fail ");
        }
    }else{
        printf(" SLS_solve exit status = %li\n", inform.status);
    }
    // obtain the solution by part solves
    for(i=0; i<n; i++) x[i] = rhs[i];
    sls_partial_solve_system( "L", &data, &status, n, x );
    sls_partial_solve_system( "D", &data, &status, n, x );
    sls_partial_solve_system( "U", &data, &status, n, x );
    sls_information( &data, &inform, &status );
    if(inform.status == 0){
        for(i=0; i<n; i++) error[i] = x[i]-sol[i];
        status = maxabsarray( error, n, &norm_residual );
        if(norm_residual < good_x){
            printf("    ok ");
        }else{
            printf("    fail ");
        }
    }else{
        printf(" SLS_solve exit status = %li\n", inform.status);
    }
    // Delete internal workspace
    sls_terminate( &data, &control, &inform );
    printf("\n");
}

int maxabsarray(real_wp_ a[], int n, real_wp_ *maxabs)
{
    int i;
    real_wp_ b, max;
    max=abs(a[0]);
    for(i=1; i<n; i++)
    {
        b = fabs(a[i]);
        if(max<b)
            max=b;
    }
    *maxabs=max;
}

```