



## C interfaces to GALAHAD LLST

Jari Fowkes and Nick Gould  
STFC Rutherford Appleton Laboratory  
Fri Jun 2 2023



<b>1 GALAHAD C package llst</b>	<b>1</b>
1.1 Introduction	1
1.1.1 Purpose	1
1.1.2 Authors	1
1.1.3 Originally released	1
1.1.4 Terminology	1
1.1.5 Method	2
1.1.6 Reference	2
1.1.7 Call order	2
1.1.8 Unsymmetric matrix storage formats	2
1.1.8.1 Dense storage format	3
1.1.8.2 Sparse co-ordinate storage format	3
1.1.8.3 Sparse row-wise storage format	3
1.1.9 Symmetric matrix storage formats	3
1.1.9.1 Dense storage format	3
1.1.9.2 Sparse co-ordinate storage format	3
1.1.9.3 Sparse row-wise storage format	3
1.1.9.4 Diagonal storage format	3
<b>2 File Index</b>	<b>5</b>
2.1 File List	5
<b>3 File Documentation</b>	<b>7</b>
3.1 galahad_llst.h File Reference	7
3.1.1 Data Structure Documentation	7
3.1.1.1 struct llst_control_type	7
3.1.1.2 struct llst_time_type	8
3.1.1.3 struct llst_history_type	9
3.1.1.4 struct llst_inform_type	9
3.1.2 Function Documentation	10
3.1.2.1 llst_initialize()	10
3.1.2.2 llst_read_specfile()	11
3.1.2.3 llst_import()	11
3.1.2.4 llst_import_scaling()	12
3.1.2.5 llst_reset_control()	13
3.1.2.6 llst_solve_problem()	14
3.1.2.7 llst_information()	16
3.1.2.8 llst_terminate()	16
<b>4 Example Documentation</b>	<b>17</b>
4.1 llstt.c	17
4.2 llsttf.c	19



# Chapter 1

## GALAHAD C package l1st

### 1.1 Introduction

#### 1.1.1 Purpose

Given a real  $m$  by  $n$  matrix  $A$ , a real  $n$  by  $n$  symmetric diagonally dominant matrix  $S$  a real  $m$  vector  $b$  and a scalar  $\Delta > 0$ , this package finds a **minimizer of the linear least-squares objective function**  $\|Ax - b\|_2$ , **where the vector  $x$  is required to satisfy the constraint**  $\|x\|_S \leq \Delta$ , where the  $S$ -norm of  $x$  is  $\|x\|_S = \sqrt{x^T S x}$ . This problem commonly occurs as a trust-region subproblem in nonlinear least-squares calculations. The package may also be used to solve the related problem in which  $x$  is instead required to satisfy the **equality constraint**  $\|x\|_S = \Delta$ . The matrix  $S$  need not be provided in the commonly-occurring  $\ell_2$ -trust-region case for which  $S = I$ , the  $n$  by  $n$  identity matrix.

Factorization of matrices of the form

$$(1) \quad K(\lambda) = \begin{pmatrix} \lambda S & A^T \\ A & -I \end{pmatrix}$$

of scalars  $\lambda$  will be required, so this package is most suited for the case where such a factorization may be found efficiently. If this is not the case, the GALAHAD package `LSTR` may be preferred.

#### 1.1.2 Authors

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

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

#### 1.1.3 Originally released

October 2008, C interface May 2023.

#### 1.1.4 Terminology

The required solution  $x_*$  necessarily satisfies the optimality condition  $A^T A x_* + \lambda_* S x_* = A^T b$ , where  $\lambda_* \geq 0$  is a Lagrange multiplier corresponding to the constraint  $\|x\|_S \leq \Delta$ ; for the equality-constrained problem  $\|x\|_S = \Delta$  and the multiplier is unconstrained.

### 1.1.5 Method

The method is iterative, and proceeds in two phases. Firstly, lower and upper bounds,  $\lambda_L$  and  $\lambda_U$ , on  $\lambda_*$  are computed using Gershgorin's theorems and other eigenvalue bounds, including those that may involve the Cholesky factorization of  $S$ . The first phase of the computation proceeds by progressively shrinking the bound interval  $[\lambda_L, \lambda_U]$  until a value  $\lambda$  for which  $\|x(\lambda)\|_M \geq \Delta$  is found. Here  $x(\lambda)$  and its companion  $y(\lambda)$  are defined to be a solution of along the way the possibility that  $\|x(0)\|_M \leq \Delta$  is examined, and if this transpires the process is terminated with  $x_* = x(0)$ . Once the terminating  $\lambda$  from the first phase has been discovered, the second phase consists of applying Newton or higher-order iterations to the nonlinear "secular" equation  $\|x(\lambda)\|_M = \Delta$  with the knowledge that such iterations are both globally and ultimately rapidly convergent.

The dominant cost is the requirement that we solve a sequence of linear systems (2). This may be rewritten as

$$(3) \quad \begin{pmatrix} \lambda S & A^T \\ A & -I \end{pmatrix} \begin{pmatrix} x(\lambda) \\ y(\lambda) \end{pmatrix} = \begin{pmatrix} A^T b \\ 0 \end{pmatrix}$$

for some auxiliary vector  $y(\lambda)$ . In general a sparse symmetric, indefinite factorization of the coefficient matrix  $K(\lambda)$  of (3) is often preferred to a Cholesky factorization of that of (2).

### 1.1.6 Reference

The method is the obvious adaptation to the linear least-squares problem of that described in detail in

H. S. Dollar, N. I. M. Gould and D. P. Robinson. On solving trust-region and other regularised subproblems in optimization. *Mathematical Programming Computation* **2(1)** (2010) 21–57.

### 1.1.7 Call order

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

- `llst_initialize` - provide default control parameters and set up initial data structures
- `llst_read_specfile` (optional) - override control values by reading replacement values from a file
- `llst_import` - set up problem data structures and fixed values
- `llst_import_scaling` (optional) - set up problem data structures for  $S$  if required
- `llst_reset_control` (optional) - possibly change control parameters if a sequence of problems are being solved
- `llst_solve_problem` - solve the trust-region problem
- `llst_information` (optional) - recover information about the solution and solution process
- `llst_terminate` - deallocate data structures

See Section 4.1 for examples of use.

### 1.1.8 Unsymmetric matrix storage formats

The unsymmetric  $m$  by  $n$  constraint matrix  $A$  may be presented and stored in a variety of convenient input formats.

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.

### 1.1.8.1 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. In this case, component  $n * i + j$  of the storage array  $A\_val$  will hold the value  $A_{ij}$  for  $0 \leq i \leq m - 1$ ,  $0 \leq j \leq n - 1$ .

### 1.1.8.2 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  $A_{ij}$ ,  $0 \leq i \leq m - 1$ ,  $0 \leq j \leq n - 1$ , are stored as the  $l$ -th components of the integer arrays  $A\_row$  and  $A\_col$  and real array  $A\_val$ , respectively, while the number of nonzeros is recorded as  $A\_ne = ne$ .

### 1.1.8.3 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  $A\_ptr$  holds the position of the first entry in this row, while  $A\_ptr(m)$  holds the total number of entries. The column indices  $j$ ,  $0 \leq j \leq n - 1$ , and values  $A_{ij}$  of the nonzero entries in the  $i$ -th row are stored in components  $l = A\_ptr(i), \dots, A\_ptr(i+1)-1$ ,  $0 \leq i \leq m - 1$ , of the integer array  $A\_col$ , and real array  $A\_val$ , respectively. For sparse matrices, this scheme almost always requires less storage than its predecessor.

## 1.1.9 Symmetric matrix storage formats

Likewise, the non-trivial symmetric  $n$  by  $n$  scaling matrix  $S$  may be presented and stored in a variety of formats. But 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).

### 1.1.9.1 Dense storage format

The matrix  $S$  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  $S$  is symmetric, only the lower triangular part (that is the part  $s_{ij}$  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  $S\_val$  will hold the value  $s_{ij}$  (and, by symmetry,  $s_{ji}$ ) 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  $S$ , its row index  $i$ , column index  $j$  and value  $s_{ij}$ ,  $0 \leq j \leq i \leq n - 1$ , are stored as the  $l$ -th components of the integer arrays  $S\_row$  and  $S\_col$  and real array  $S\_val$ , respectively, while the number of nonzeros is recorded as  $S\_ne = ne$ . 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  $S$  the  $i$ -th component of the integer array  $S\_ptr$  holds the position of the first entry in this row, while  $S\_ptr(n)$  holds the total number of entries. The column indices  $j$ ,  $0 \leq j \leq i$ , and values  $s_{ij}$  of the entries in the  $i$ -th row are stored in components  $l = S\_ptr(i), \dots, S\_ptr(i+1)-1$  of the integer array  $S\_col$ , and real array  $S\_val$ , 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.

### 1.1.9.4 Diagonal storage format

If  $S$  is diagonal (i.e.,  $s_{ij} = 0$  for all  $0 \leq i \neq j \leq n - 1$ ) only the diagonal entries  $s_{ii}$ ,  $0 \leq i \leq n - 1$  need be stored, and the first  $n$  components of the array  $S\_val$  may be used for the purpose.





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">galahad_llst.h</a> . . . . .	7
--	---



## Chapter 3

# File Documentation

### 3.1 galahad\_llst.h File Reference

```
#include <stdbool.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sb1s.h"
#include "galahad_sls.h"
#include "galahad_ir.h"
```

#### Data Structures

- struct [llst\\_control\\_type](#)
- struct [llst\\_time\\_type](#)
- struct [llst\\_history\\_type](#)
- struct [llst\\_inform\\_type](#)

#### Functions

- void [llst\\_initialize](#) (void \*\*data, struct [llst\\_control\\_type](#) \*control, int \*status)
- void [llst\\_read\\_specfile](#) (struct [llst\\_control\\_type](#) \*control, const char specfile[])
- void [llst\\_import](#) (struct [llst\\_control\\_type](#) \*control, void \*\*data, int \*status, int m, int n, const char A\_type[], int A\_ne, const int A\_row[], const int A\_col[], const int A\_ptr[])
- void [llst\\_import\\_scaling](#) (struct [llst\\_control\\_type](#) \*control, void \*\*data, int \*status, int n, const char S\_type[], int S\_ne, const int S\_row[], const int S\_col[], const int S\_ptr[])
- void [llst\\_reset\\_control](#) (struct [llst\\_control\\_type](#) \*control, void \*\*data, int \*status)
- void [llst\\_solve\\_problem](#) (void \*\*data, int \*status, int m, int n, const real\_wp\_ radius, int A\_ne, const real\_wp\_ A\_val[], const real\_wp\_ b[], real\_wp\_ x[], int S\_ne, const real\_wp\_ S\_val[])↔
- void [llst\\_information](#) (void \*\*data, struct [llst\\_inform\\_type](#) \*inform, int \*status)
- void [llst\\_terminate](#) (void \*\*data, struct [llst\\_control\\_type](#) \*control, struct [llst\\_inform\\_type](#) \*inform)

#### 3.1.1 Data Structure Documentation

##### 3.1.1.1 struct llst\_control\_type

control derived type as a C struct

##### Examples

[llst.c](#), and [llstf.c](#).

## Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	unit for error messages
int	out	unit for monitor output
int	print_level	controls level of diagnostic output
int	new_a	how much of $A$ has changed since the previous call. Possible values are <ul style="list-style-type: none"> <li>• 0 unchanged</li> <li>• 1 values but not indices have changed</li> <li>• 2 values and indices have changed</li> </ul>
int	new_s	how much of $S$ has changed since the previous call. Possible values are <ul style="list-style-type: none"> <li>• 0 unchanged</li> <li>• 1 values but not indices have changed</li> <li>• 2 values and indices have changed</li> </ul>
int	max_factorizations	the maximum number of factorizations (=iterations) allowed. -ve implies no limit
int	taylor_max_degree	maximum degree of Taylor approximant allowed ( $\leq 3$ )
real_wp_	initial_multiplier	initial estimate of the Lagrange multiplier
real_wp_	lower	lower and upper bounds on the multiplier, if known
real_wp_	upper	see lower
real_wp_	stop_normal	stop when $  x   - \text{radius} \leq \max(\text{stop\_normal} * \max(1, \text{radius}))$
bool	equality_problem	is the solution is $<b<$ required to lie on the boundary (i.e., is the constraint an equality)?
bool	use_initial_multiplier	ignore initial_multiplier?
bool	space_critical	if space is critical, ensure allocated arrays are no bigger than needed
bool	deallocate_error_fatal	exit if any deallocation fails
char	definite_linear_solver[31]	definite linear equation solver
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'
struct sbls_control_type	sbls_control	control parameters for the symmetric factorization and related linear solves (see sbls_c documentation)
struct sls_control_type	sls_control	control parameters for the factorization of $S$ and related linear solves (see sls_c documentation)
struct ir_control_type	ir_control	control parameters for iterative refinement for definite system solves (see ir_c documentation)

## 3.1.1.2 struct llst\_time\_type

time derived type as a C struct

## Data Fields

real_wp_	total	total CPU time spent in the package
real_wp_	assemble	CPU time assembling $K(\lambda)$ in (1)
real_wp_	analyse	CPU time spent analysing $K(\lambda)$ .
real_wp_	factorize	CPU time spent factorizing $K(\lambda)$ .
real_wp_	solve	CPU time spent solving linear systems involving $K(\lambda)$ .
real_wp_	clock_total	total clock time spent in the package
real_wp_	clock_assemble	clock time assembling $K(\lambda)$
real_wp_	clock_analyse	clock time spent analysing $K(\lambda)$
real_wp_	clock_factorize	clock time spent factorizing $K(\lambda)$
real_wp_	clock_solve	clock time spent solving linear systems involving $K(\lambda)$

## 3.1.1.3 struct llst\_history\_type

history derived type as a C struct

## Data Fields

real_wp_	lambda	the value of $\lambda$
real_wp_	x_norm	the corresponding value of $\ x(\lambda)\ _M$
real_wp_	r_norm	the corresponding value of $\ Ax(\lambda) - b\ _2$

## 3.1.1.4 struct llst\_inform\_type

inform derived type as a C struct

## Examples

[llst.c](#), and [llstf.c](#).

## Data Fields

int	status	reported return status: <ul style="list-style-type: none"> <li>• 0 the solution has been found</li> <li>• -1 an array allocation has failed</li> <li>• -2 an array deallocation has failed</li> <li>• -3 n and/or Delta is not positive</li> <li>• -10 the factorization of <math>K(\lambda)</math> failed</li> <li>• -15 <math>S</math> does not appear to be strictly diagonally dominant</li> <li>• -16 ill-conditioning has prevented further progress</li> </ul>
int	alloc_status	STAT value after allocate failure.

## Data Fields

int	factorizations	the number of factorizations performed
int	len_history	the number of $(\ x\ _S, \lambda)$ pairs in the history
real_wp_	r_norm	corresponding value of the two-norm of the residual, $\ Ax(\lambda) - b\ $
real_wp_	x_norm	the S-norm of $x$ , $\ x\ _S$
real_wp_	multiplier	the Lagrange multiplier corresponding to the trust-region constraint
char	bad_alloc[81]	name of array which provoked an allocate failure
struct <a href="#">llst_time_type</a>	time	time information
struct <a href="#">llst_history_type</a>	history[100]	history information
struct <a href="#">sbls_inform_type</a>	sbls_inform	information from the symmetric factorization and related linear solves (see <a href="#">sbls_c</a> documentation)
struct <a href="#">sls_inform_type</a>	sls_inform	information from the factorization of $S$ and related linear solves (see <a href="#">sls_c</a> documentation)
struct <a href="#">ir_inform_type</a>	ir_inform	information from the iterative refinement for definite system solves (see <a href="#">ir_c</a> documentation)

## 3.1.2 Function Documentation

3.1.2.1 [llst\\_initialize\(\)](#)

```
void llst_initialize (
    void ** data,
    struct llst\_control\_type * control,
    int * status )
```

Set default control values and initialize private data

## Parameters

in, out	<i>data</i>	holds private internal data
out	<i>control</i>	is a struct containing control information (see <a href="#">llst_control_type</a> )
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"> <li>• 0. The import was succesful.</li> </ul>

## Examples

[llst.c](#), and [llstf.c](#).

## 3.1.2.2 llst\_read\_specfile()

```
void llst_read_specfile (
    struct llst_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

## Parameters

in, out	<i>control</i>	is a struct containing control information (see <a href="#">llst_control_type</a> )
in	<i>specfile</i>	is a character string containing the name of the specification file

## 3.1.2.3 llst\_import()

```
void llst_import (
    struct llst_control_type * control,
    void ** data,
    int * status,
    int m,
    int n,
    const char A_type[],
    int A_ne,
    const int A_row[],
    const int A_col[],
    const int A_ptr[] )
```

Import problem data into internal storage prior to solution.

## Parameters

in	<i>control</i>	is a struct whose members provide control paramters for the remaining prcedures (see <a href="#">llst_control_type</a> )
in, out	<i>data</i>	holds private internal data
in, 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"> <li>• 1. The import was succesful, and the package is ready for the solve phase</li> <li>• -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.</li> <li>• -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.</li> <li>• -3. The restriction <math>n &gt; 0</math> or requirement that type contains its relevant string 'dense', 'coordinate', 'sparse_by_rows', 'diagonal' or 'absent' has been violated.</li> </ul>

## Parameters

in	<i>m</i>	is a scalar variable of type int, that holds the number of residuals, i.e., the number of rows of <i>A</i> . <i>m</i> must be positive.
in	<i>n</i>	is a scalar variable of type int, that holds the number of variables, i.e., the number of columns of <i>A</i> . <i>n</i> must be positive.
in	<i>A_type</i>	is a one-dimensional array of type char that specifies the <a href="#">unsymmetric storage scheme</a> used for the constraint Jacobian, <i>A</i> if any. It should be one of 'coordinate', 'sparse_by_rows' or 'dense'; lower or upper case variants are allowed.
in	<i>A_ne</i>	is a scalar variable of type int, that holds the number of entries in <i>A</i> , if used, in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	<i>A_row</i>	is a one-dimensional array of size <i>A_ne</i> and type int, that holds the row indices of <i>A</i> in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes, and in this case can be NULL.
in	<i>A_col</i>	is a one-dimensional array of size <i>A_ne</i> and type int, that holds the column indices of <i>A</i> in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense or diagonal storage schemes are used, and in this case can be NULL.
in	<i>A_ptr</i>	is a one-dimensional array of size <i>n</i> +1 and type int, that holds the starting position of each row of <i>A</i> , 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 NULL.

## Examples

[llst.c](#), and [llstf.c](#).

## 3.1.2.4 llst\_import\_scaling()

```
void llst_import_scaling (
    struct llst_control_type * control,
    void ** data,
    int * status,
    int n,
    const char S_type[],
    int S_ne,
    const int S_row[],
    const int S_col[],
    const int S_ptr[] )
```

Import the scaling matrix *S* into internal storage prior to solution. Thus must have been preceded by a call to `llst_import`.

## Parameters

in	<i>control</i>	is a struct whose members provide control paramters for the remaining procedures (see <a href="#">llst_control_type</a> )
in, out	<i>data</i>	holds private internal data



## Parameters

in, 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"> <li>• 1. The import was succesful, and the package is ready for the solve phase</li> <li>• -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.</li> <li>• -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.</li> <li>• -3. The restriction <math>n &gt; 0</math> or requirement that type contains its relevant string 'dense', 'coordinate', 'sparse_by_rows' or 'diagonal' has been violated.</li> </ul>
in	<i>n</i>	is a scalar variable of type int, that holds the number of variables, i.e., the number of rows and columns of $S$ . $n$ must be positive.
in	<i>S_type</i>	is a one-dimensional array of type char that specifies the symmetric storage scheme used for the matrix $S$ . It should be one of 'coordinate', 'sparse_by_rows', 'dense' or 'diagonal'; lower or upper case variants are allowed.
in	<i>S_ne</i>	is a scalar variable of type int, that holds the number of entries in the lower triangular part of $S$ in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	<i>S_row</i>	is a one-dimensional array of size <i>S_ne</i> and type int, that holds the row indices of the lower triangular part of $S$ 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 NULL.
in	<i>S_col</i>	is a one-dimensional array of size <i>S_ne</i> and type int, that holds the column indices of the lower triangular part of $S$ in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense, diagonal or (scaled) identity storage schemes are used, and in this case can be NULL.
in	<i>S_ptr</i>	is a one-dimensional array of size $n+1$ and type int, that holds the starting position of each row of the lower triangular part of $S$ , 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 NULL.

## Examples

[llst.c](#), and [llstf.c](#).

## 3.1.2.5 llst\_reset\_control()

```
void llst_reset_control (
    struct llst_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 <a href="#">llst_control_type</a> )
in, out	<i>data</i>	holds private internal data
in, 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"> <li>1. The import was succesful, and the package is ready for the solve phase</li> </ul>

## 3.1.2.6 llst\_solve\_problem()

```

void llst_solve_problem (
    void ** data,
    int * status,
    int m,
    int n,
    const real_wp_ radius,
    int A_ne,
    const real_wp_ A_val[],
    const real_wp_ b[],
    real_wp_ x[],
    int S_ne,
    const real_wp_ S_val[] )

```

Solve the trust-region problem.

## Parameters

in, out	<i>data</i>	holds private internal data
---------	-------------	-----------------------------

## Parameters

<i>in, out</i>	<i>status</i>	<p>is a scalar variable of type int, that gives the entry and exit status from the package. Possible exit are:</p> <ul style="list-style-type: none"> <li>• 0. The run was succesful.</li> <li>• -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.</li> <li>• -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.</li> <li>• -3. The restrictions <math>n &gt; 0</math> and <math>m &gt; 0</math> or requirement that <i>A_type</i> or <i>A_type</i> contains its relevant string 'dense', 'coordinate', 'sparse_by_rows' or 'diagonal' has been violated.</li> <li>• -9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component inform.factor_status</li> <li>• -10. The factorization failed; the return status from the factorization package is given in the component inform.factor_status.</li> <li>• -11. The solution of a set of linear equations using factors from the factorization package failed; the return status from the factorization package is given in the component inform.factor_status.</li> <li>• -15. The matrix <math>S</math> does not appear to be strictly diagonally dominant.</li> <li>• -16. The problem is so ill-conditioned that further progress is impossible.</li> <li>• -17. The step is too small to make further impact.</li> </ul>
<i>in</i>	<i>m</i>	is a scalar variable of type int, that holds the number of residuals
<i>in</i>	<i>n</i>	is a scalar variable of type int, that holds the number of variables
<i>in</i>	<i>radius</i>	is a scalar of type double, that holds the trust-region radius, $\Delta$ , used. radius must be strictly positive
<i>in</i>	<i>A_ne</i>	is a scalar variable of type int, that holds the number of entries in the observation matrix $A$ .
<i>in</i>	<i>A_val</i>	is a one-dimensional array of size <i>A_ne</i> and type double, that holds the values of the entries of the observation matrix $A$ in any of the available storage schemes.
<i>in</i>	<i>b</i>	is a one-dimensional array of size <i>m</i> and type double, that holds the values $b$ of observations. The <i>i</i> -th component of <i>b</i> , $i = 0, \dots, m-1$ , contains $b_i$ .
<i>out</i>	<i>x</i>	is a one-dimensional array of size <i>n</i> and type double, that holds the values $x$ of the optimization variables. The <i>j</i> -th component of <i>x</i> , $j = 0, \dots, n-1$ , contains $x_j$ .
<i>in</i>	<i>S_ne</i>	is a scalar variable of type int, that holds the number of entries in the scaling matrix $S$ if it not the identity matrix.
<i>in</i>	<i>S_val</i>	is a one-dimensional array of size <i>S_ne</i> and type double, that holds the values of the entries of the scaling matrix $S$ in any of the available storage schemes. If <i>S_val</i> is NULL, $S$ will be taken to be the identity matrix.

## Examples

[llst.c](#), and [llstf.c](#).

### 3.1.2.7 llst\_information()

```
void llst_information (
    void ** data,
    struct llst_inform_type * inform,
    int * status )
```

Provides output information

#### Parameters

in, out	<i>data</i>	holds private internal data
out	<i>inform</i>	is a struct containing output information (see <a href="#">llst_inform_type</a> )
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"> <li>• 0. The values were recorded succesfully</li> </ul>

#### Examples

[llstt.c](#), and [llsttf.c](#).

### 3.1.2.8 llst\_terminate()

```
void llst_terminate (
    void ** data,
    struct llst_control_type * control,
    struct llst_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 <a href="#">llst_control_type</a> )
out	<i>inform</i>	is a struct containing output information (see <a href="#">llst_inform_type</a> )

#### Examples

[llstt.c](#), and [llsttf.c](#).

## Chapter 4

# Example Documentation

### 4.1 llstf.c

This is an example of how to use the package.

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

```
/* llstf.c */
/* Full test for the LLST C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_llst.h"
int main(void) {
    // Derived types
    void *data;
    struct llst_control_type control;
    struct llst_inform_type inform;
    int i, l;
    // Set problem data
    // set dimensions
    int m = 100;
    int n = 2*m+1;
    // A = ( I : Diag(1:n) : e )
    int A_ne = 3*m;
    int A_row[A_ne];
    int A_col[A_ne];
    int A_ptr[m+1];
    real_wp A_val[A_ne];
    // store A in sparse formats
    l=0;
    for( i=0; i < m; i++){
        A_ptr[i] = l;
        A_row[l] = i;
        A_col[l] = i;
        A_val[l] = 1.0;
        l++;
        A_row[l] = i;
        A_col[l] = m+i;
        A_val[l] = i;
        l++;
        A_row[l] = i;
        A_col[l] = n-1;
        A_val[l] = 1.0;
        l++;
    }
    A_ptr[m] = l;
    // store A in dense format
    int A_dense_ne = m * n;
    real_wp A_dense_val[A_dense_ne];
    for( i=0; i < A_dense_ne; i++) A_dense_val[i] = 0.0;
    l=-1;
    for( i=1; i <= m; i++){
        A_dense_val[l+i] = 1.0;
        A_dense_val[l+m+i] = i;
```

```

    A_dense_val[l+n] = 1.0;
    l=l+n;
}
// S = diag(1:n)**2
int S_ne = n;
int S_row[S_ne];
int S_col[S_ne];
int S_ptr[n+1];
real_wp_ S_val[S_ne];
// store S in sparse formats
for( i=0; i < n; i++){
    S_row[i] = i;
    S_col[i] = i;
    S_ptr[i] = i;
    S_val[i] = (i+1)*(i+1);
}
S_ptr[n] = n;
// store S in dense format
int S_dense_ne = n*(n+1)/2;
real_wp_ S_dense_val[S_dense_ne];
for( i=0; i < S_dense_ne; i++) S_dense_val[i] = 0.0;
l=-1;
for( i=1; i <= n; i++){
    S_dense_val[l+i] = i*i;
    l=l+i;
}
// b is a vector of ones
real_wp_ b[m]; // observations
for( i=0; i < m; i++){
    b[i] = 1.0;
}
// trust-region radius is one
real_wp_ radius = 1.0;
// Set output storage
real_wp_ x[n]; // solution
char st;
int status;
printf(" C sparse matrix indexing\n\n");
printf(" basic tests of problem storage formats\n\n");
// loop over storage formats
for( int d=1; d<=4; d++){
    // Initialize LLST
    llst_initialize( &data, &control, &status );
    strcpy(control.definite_linear_solver, "potr " );
    strcpy(control.sbls_control.symmetric_linear_solver, "sytr " );
    strcpy(control.sbls_control.definite_linear_solver, "potr " );
    // control.print_level = 1;
    // Set user-defined control options
    control.f_indexing = false; // C sparse matrix indexing
    // use s or not (1 or 0)
    for( int use_s=0; use_s<=1; use_s++){
        switch(d){
            case 1: // sparse co-ordinate storage
                st = 'C';
                llst_import( &control, &data, &status, m, n,
                    "coordinate", A_ne, A_row, A_col, NULL );
                if(use_s == 0){
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, 0, NULL );
                }else{
                    llst_import_scaling( &control, &data, &status, n,
                        "coordinate", S_ne, S_row,
                        S_col, NULL );
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, S_ne, S_val );
                }
                break;
            case 2: // sparse by rows
                st = 'R';
                llst_import( &control, &data, &status, m, n,
                    "sparse_by_rows", A_ne, NULL, A_col, A_ptr );
                if(use_s == 0){
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, 0, NULL );
                }else{
                    llst_import_scaling( &control, &data, &status, n,
                        "sparse_by_rows", S_ne, NULL,
                        S_col, S_ptr );
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, S_ne, S_val );
                }
                break;
            case 3: // dense
                st = 'D';
                llst_import( &control, &data, &status, m, n,
                    "dense", A_dense_ne, NULL, NULL, NULL );
                if(use_s == 0){

```

```

        llst_solve_problem( &data, &status, m, n, radius,
                           A_dense_ne, A_dense_val, b, x,
                           0, NULL );
    }else{
        llst_import_scaling( &control, &data, &status, n,
                             "dense", S_dense_ne,
                             NULL, NULL, NULL );
        llst_solve_problem( &data, &status, m, n, radius,
                           A_dense_ne, A_dense_val, b, x,
                           S_dense_ne, S_dense_val );
    }
    break;
case 4: // diagonal
    st = 'I';
    llst_import( &control, &data, &status, m, n,
                "coordinate", A_ne, A_row, A_col, NULL );
    if(use_s == 0){
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, 0, NULL );
    }else{
        llst_import_scaling( &control, &data, &status, n,
                             "diagonal", S_ne, NULL, NULL, NULL );
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, S_ne, S_val );
    }
    break;
}
llst_information( &data, &inform, &status );
if(inform.status == 0){
    printf("storage type %c%i: status = %li, ||r|| = %5.2f\n",
          st, use_s, inform.status, inform.r_norm );
}else{
    printf("storage type %c%i: LLST_solve exit status = %li\n",
          st, use_s, inform.status);
}
}
//printf("x: ");
//for( int i = 0; i < n; i++) printf("%f ", x[i]);
//printf("\n");
// Delete internal workspace
llst_terminate( &data, &control, &inform );
}
}

```

## 4.2 llsttf.c

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

```

/* llsttf.c */
/* Full test for the LLST C interface using Fortran sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_llst.h"
int main(void) {
    // Derived types
    void *data;
    struct llst_control_type control;
    struct llst_inform_type inform;
    int i, l;
    // Set problem data
    // set dimensions
    int m = 100;
    int n = 2*m+1;
    // A = ( I : Diag(1:n) : e )
    int A_ne = 3*m;
    int A_row[A_ne];
    int A_col[A_ne];
    int A_ptr[m+1];
    real_wp_ A_val[A_ne];
    // store A in sparse formats
    l=0;
    for( i=1; i <= m; i++){
        A_ptr[i-1] = l+1;
        A_row[l] = i;
        A_col[l] = i;
        A_val[l] = 1.0;
        l++;
        A_row[l] = i;
    }
}

```

```

    A_col[l] = m+i;
    A_val[l] = i;
    l++;
    A_row[l] = i;
    A_col[l] = n;
    A_val[l] = 1.0;
    l++;
}
A_ptr[m] = l+1;
// store A in dense format
int A_dense_ne = m * n;
real_wp_ A_dense_val[A_dense_ne];
for( i=0; i < A_dense_ne; i++) A_dense_val[i] = 0.0;
l=-1;
for( i=1; i <= m; i++){
    A_dense_val[l+i] = 1.0;
    A_dense_val[l+m+i] = i;
    A_dense_val[l+n] = 1.0;
    l=l+n;
}
// S = diag(1:n)**2
int S_ne = n;
int S_row[S_ne];
int S_col[S_ne];
int S_ptr[n+1];
real_wp_ S_val[S_ne];
// store S in sparse formats
for( i=0; i < n; i++){
    S_row[i] = i+1;
    S_col[i] = i+1;
    S_ptr[i] = i+1;
    S_val[i] = (i+1)*(i+1);
}
S_ptr[n] = n+1;
// store S in dense format
int S_dense_ne = n*(n+1)/2;
real_wp_ S_dense_val[S_dense_ne];
for( i=0; i < S_dense_ne; i++) S_dense_val[i] = 0.0;
l=-1;
for( i=1; i <= n; i++){
    S_dense_val[l+i] = i*i;
    l=l+i;
}
// b is a vector of ones
real_wp_ b[m]; // observations
for( i=0; i < m; i++){
    b[i] = 1.0;
}
// trust-region radius is one
real_wp_ radius = 1.0;
// Set output storage
real_wp_ x[n]; // solution
char st;
int status;
printf(" Fortran sparse matrix indexing\n\n");
printf(" basic tests of problem storage formats\n\n");
// loop over storage formats
for( int d=1; d<=4; d++){
    // Initialize LLST
    llst_initialize( &data, &control, &status );
    strcpy(control.definite_linear_solver, "potr " );
    strcpy(control.sbls_control.symmetric_linear_solver, "sytr " );
    strcpy(control.sbls_control.definite_linear_solver, "potr " );
    // control.print_level = 1;
    // Set user-defined control options
    control.f_indexing = true; // Fortran sparse matrix indexing
    // use s or not (1 or 0)
    for( int use_s=0; use_s<=1; use_s++){
        switch(d){
            case 1: // sparse co-ordinate storage
                st = 'C';
                llst_import( &control, &data, &status, m, n,
                    "coordinate", A_ne, A_row, A_col, NULL );
                if(use_s == 0){
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, 0, NULL );
                }else{
                    llst_import_scaling( &control, &data, &status, n,
                        "coordinate", S_ne, S_row,
                        S_col, NULL );
                    llst_solve_problem( &data, &status, m, n, radius,
                        A_ne, A_val, b, x, S_ne, S_val );
                }
                break;
            case 2: // sparse by rows
                st = 'R';
                llst_import( &control, &data, &status, m, n,

```



```

        "sparse_by_rows", A_ne, NULL, A_col, A_ptr );
    if(use_s == 0){
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, 0, NULL );
    }else{
        llst_import_scaling( &control, &data, &status, n,
                           "sparse_by_rows", S_ne, NULL,
                           S_col, S_ptr );
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, S_ne, S_val );
    }
    break;
case 3: // dense
    st = 'D';
    llst_import( &control, &data, &status, m, n,
               "dense", A_dense_ne, NULL, NULL, NULL );
    if(use_s == 0){
        llst_solve_problem( &data, &status, m, n, radius,
                           A_dense_ne, A_dense_val, b, x,
                           0, NULL );
    }else{
        llst_import_scaling( &control, &data, &status, n,
                           "dense", S_dense_ne,
                           NULL, NULL, NULL );
        llst_solve_problem( &data, &status, m, n, radius,
                           A_dense_ne, A_dense_val, b, x,
                           S_dense_ne, S_dense_val );
    }
    break;
case 4: // diagonal
    st = 'I';
    llst_import( &control, &data, &status, m, n,
               "coordinate", A_ne, A_row, A_col, NULL );
    if(use_s == 0){
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, 0, NULL );
    }else{
        llst_import_scaling( &control, &data, &status, n,
                           "diagonal", S_ne, NULL, NULL, NULL );
        llst_solve_problem( &data, &status, m, n, radius,
                           A_ne, A_val, b, x, S_ne, S_val );
    }
    break;
}
llst_information( &data, &inform, &status );
if(inform.status == 0){
    printf("storage type %c%i: status = %li, ||r|| = %5.2f\n",
           st, use_s, inform.status, inform.r_norm );
}else{
    printf("storage type %c%i: LLST_solve exit status = %li\n",
           st, use_s, inform.status);
}
}
//printf("x: ");
//for( int i = 0; i < n; i++) printf("%f ", x[i]);
//printf("\n");
// Delete internal workspace
llst_terminate( &data, &control, &inform );
}

```

