C interfaces to GALAHAD LSTR

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Mon Feb 21 2022

Chapter 1

GALAHAD C package lstr

1.1 Introduction

1.1.1 Purpose

Given a real m by n matrix A, a real m vector b and a scalar $\Delta > 0$, this package finds an **approximate minimizer of $\|Ax - b\|_2$, where the vector x is required to satisfy the ``trust-region'' constraint $\|x\|_2 \leq \Delta$.** This problem commonly occurs as a trust-region subproblem in nonlinear optimization calculations, and may be used to regularize the solution of under-determined or ill-conditioned linear least-squares problems. The method may be suitable for large m and/or n as no factorization involving A is required. Reverse communication is used to obtain matrix-vector products of the form $u + Av$ and $v + A^T u$.

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

November 2007, C interface December 2021.

1.1.4 Terminology

The required solution x necessarily satisfies the optimality condition $A^T(Ax - b) + \lambda x = 0$, where $\lambda \geq 0$ is a Lagrange multiplier corresponding to the trust-region constraint $\|x\|_2 \leq \Delta$.

1.1.5 Method

The method is iterative. Starting with the vector $u_1 = b$, a bi-diagonalisation process is used to generate the vectors v_k and $u_k + 1$ so that the n by k matrix $V_k = (v_1 \ldots v_k)$ and the m by $(k+1)$ matrix $U_k = (u_1 \ldots u_{k+1})$ together satisfy

$$AV_k = U_{k+1}B_k \text{ and } b = \|b\|U_{k+1}e_1,$$

where B_k is $(k+1)$ by k and lower bi-diagonal, U_k and V_k have orthonormal columns and e_1 is the first unit vector. The solution sought is of the form $x_k = V_k y_k$, where y_k solves the bi-diagonal least-squares trust-region problem

$$(1) \quad \min \|B_k y - \|b\|e_1\|_2 \text{ subject to } \|y\|_2 \leq \Delta.$$

If the trust-region constraint is inactive, the solution y_k may be found, albeit indirectly, via the LSQR algorithm of Paige and Saunders which solves the bi-diagonal least-squares problem

$$\min \|B_k y - \|b\|e_1\|_2$$

using a QR factorization of B_k. Only the most recent v_k and u_{k+1} are required, and their predecessors discarded, to compute x_k from x_{k-1}. This method has the important property that the iterates y (and thus x_k) generated increase in norm with k. Thus as soon as an LSQR iterate lies outside the trust-region, the required solution to (1) and thus to the original problem must lie on the boundary of the trust-region.

If the solution is so constrained, the simplest strategy is to interpolate the last interior iterate with the newly discovered exterior one to find the boundary point—the so-called Steihaug-Toint point—between them. Once the solution is known to lie on the trust-region boundary, further improvement may be made by solving

$$\min \|B_k y - \|b\|e_1\|_2 \text{ subject to } |\|y\|_2 = \Delta,$$

for which the optimality conditions require that $y_k = y(\lambda_k)$ where λ_k is the positive root of

$$B_k^T (B_k y(\lambda) - \|b\|e_1) + \lambda y(\lambda) = 0 \text{ and } \|y(\lambda)\|_2 = \Delta$$

The vector $y(\lambda)$ is equivalently the solution to the regularized least-squares problem

$$\min \left\| \begin{pmatrix} B_k \\ \lambda^{\frac{1}{2}} I \end{pmatrix} y - \|b\|e_1 \right\|$$

and may be found efficiently. Given $y(\lambda)$, Newton's method is then used to find λ_k as the positive root of $\|y(\lambda)\|_2 = \Delta$. Unfortunately, unlike when the solution lies in the interior of the trust-region, it is not known how to recur x_k from x_{k-1} given y_k, and a second pass in which $x_k = V_k y_k$ is regenerated is needed—this need only be done once x_k has implicitly deemed to be sufficiently close to optimality. As this second pass is an additional expense, a record is kept of the optimal objective function values for each value of k, and the second pass is only performed so far as to ensure a given fraction of the final optimal objective value. Large savings may be made in the second pass by choosing the required fraction to be significantly smaller than one.

1.1.6 Reference

A complete description of the unconstrained case is given by

C. C. Paige and M. A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software, 8(1):43–71, 1982

and

C. C. Paige and M. A. Saunders, ALGORITHM 583: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software, 8(2):195–209, 1982.

Additional details on how to proceed once the trust-region constraint is encountered are described in detail in

C. Cartis, N. I. M. Gould and Ph. L. Toint, Trust-region and other regularisation of linear least-squares problems. BIT 49(1):21-53 (2009).

1.1.7 Call order

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

- lstr_initialize - provide default control parameters and set up initial data structures
- lstr_read_specfile (optional) - override control values by reading replacement values from a file
- lstr_import_control - import control parameters prior to solution
- lstr_solve_problem - solve the problem by reverse communication, a sequence of calls are made under control of a status parameter, each exit either asks the user to provide additional informaton and to re-enter, or reports that either the solution has been found or that an error has occurred
- lstr_information (optional) - recover information about the solution and solution process
- lstr_terminate - deallocate data structures

See Section 4.1 for an example of use.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 lstr.h File Reference

```
#include <stdbool.h>
#include "galahad_precision.h"
```

Data Structures

- struct lstr_control_type
- struct lstr_inform_type

Functions

- void lstr_initialize (void **data, struct lstr_control_type *control, int *status)
- void lstr_read_specfile (struct lstr_control_type *control, const char specfile[ ])
- void lstr_import_control (struct lstr_control_type *control, void **data, int *status)
- void lstr_solve_problem (void **data, int *status, int m, int n, const real_wp_ radius, real_wp_ x[ ], real_wp_ u[ ], real_wp_ v[ ])
- void lstr_information (void **data, struct lstr_inform_type *inform, int *status)
- void lstr_terminate (void **data, struct lstr_control_type *control, struct lstr_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct lstr_control_type

control derived type as a C struct

Examples

lstrt.c.

Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	error and warning diagnostics occur on stream error
int	out	general output occurs on stream out
int	print_level	the level of output required is specified by print_level
int	start_print	any printing will start on this iteration
int	stop_print	any printing will stop on this iteration
int	print_gap	the number of iterations between printing
int	itmin	the minimum number of iterations allowed (-ve = no bound)
int	itmax	the maximum number of iterations allowed (-ve = no bound)
int	itmax_on_boundary	the maximum number of iterations allowed once the boundary has been encountered (-ve = no bound)
int	bitmax	the maximum number of Newton inner iterations per outer iteration allowe (-ve = no bound)
int	extra_vectors	the number of extra work vectors of length n used
real_wp_	stop_relative	the iteration stops successfully when $\|A^T r\|$ is less than max(stop_relative $* \|A^T r_{initial}\|$, stop_absolute)
real_wp_	stop_absolute	see stop_relative
real_wp_	fraction_opt	an estimate of the solution that gives at least .fraction_opt times the optimal objective value will be found
real_wp_	time_limit	the maximum elapsed time allowed (-ve means infinite)
bool	steihaug_toint	should the iteration stop when the Trust-region is first encountered?
bool	space_critical	if .space_critical true, every effort will be made to use as little space as possible. This may result in longer computation time
bool	deallocate_error_fatal	if .deallocate_error_fatal is true, any array/pointer deallocation error will terminate execution. Otherwise, computation will continue
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 lstr_inform_type

inform derived type as a C struct

Examples

lstrt.c.

Data Fields

int	status	return status. See lstr_solve_problem for details
int	alloc_status	the status of the last attempted allocation/deallocation
char	bad_alloc[81]	the name of the array for which an allocation/deallocation error ocurred
int	iter	the total number of iterations required
int	iter_pass2	the total number of pass-2 iterations required if the solution lies on the trust-region boundary
int	biters	the total number of inner iterations performed
int	biter_min	the smallest number of inner iterations performed during an outer iteration

Data Fields

int	biter_max	the largestt number of inner iterations performed during an outer iteration
real_wp_	multiplier	the Lagrange multiplier, λ, corresponding to the trust-region constraint
real_wp_	x_norm	the Euclidean norm of x
real_wp_	r_norm	the Euclidean norm of $Ax - b$
real_wp_	Atr_norm	the Euclidean norm of $A^T(Ax - b) + \lambda x$
real_wp_	biter_mean	the average number of inner iterations performed during an outer iteration

3.1.2 Function Documentation

3.1.2.1 lstr_initialize()

```
void lstr_initialize (
            void ** data,
            struct lstr_control_type * control,
            int * status )
```

Set default control values and initialize private data

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see lstr_control_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The import was succesful.

Examples

lstrt.c.

3.1.2.2 lstr_read_specfile()

```
void lstr_read_specfile (
            struct lstr_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	*control*	is a struct containing control information (see lstr_control_type)
in	*specfile*	is a character string containing the name of the specification file

3.1.2.3 lstr_import_control()

```
void lstr_import_control (
            struct lstr_control_type * control,
            void ** data,
            int * status )
```

Import control parameters prior to solution.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see lstr_control_type)
in,out	*data*	holds private internal data
in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 1. The import was succesful, and the package is ready for the solve phase

Examples

lstrt.c.

3.1.2.4 lstr_solve_problem()

```
void lstr_solve_problem (
            void ** data,
            int * status,
            int m,
            int n,
            const real_wp_ radius,
            real_wp_ x[ ],
            real_wp_ u[ ],
            real_wp_ v[ ] )
```

Solve the trust-region least-squares problem using reverse communication.

Parameters

in,out	*data*	holds private internal data

Parameters

<table>
<tr><td>in,out</td><td>status</td><td>is a scalar variable of type int, that gives the entry and exit status from the package. This must be set to
1. on initial entry. Set u (below) to b for this entry.
5. the iteration is to be restarted with a smaller radius but with all other data unchanged. Set u (below) to b for this entry.
Possible exit values are:
0. the solution has been found
2. The user must perform the operation $$u := u + Av,$$ and recall the function. The vectors u and v are available in the arrays u and v (below) respectively, and the result u must overwrite the content of u. No argument except u should be altered before recalling the function
3. The user must perform the operation $$v := v + A^T u,$$ and recall the function. The vectors u and v are available in the arrays u and v (below) respectively, and the result v must overwrite the content of v. No argument except v should be altered before recalling the function
4. The user must reset u (below) to b are recall the function. No argument except u should be altered before recalling the function
-1. an array allocation has failed
-2. an array deallocation has failed
-3. one or more of n, m or weight violates allowed bounds
-18. the iteration limit has been exceeded
-25. status is negative on entry</td></tr>
<tr><td>in</td><td>m</td><td>is a scalar variable of type int, that holds the number of equations (i.e., rows of A), $m > 0$</td></tr>
<tr><td>in</td><td>n</td><td>is a scalar variable of type int, that holds the number of variables (i.e., columns of A), $n > 0$</td></tr>
<tr><td>in</td><td>radius</td><td>is a scalar of type double, that holds the trust-region radius, $\Delta > 0$</td></tr>
<tr><td>in,out</td><td>x</td><td>is a one-dimensional array of size n and type double, that holds the solution x. The j-th component of x, j = 0, ... , n-1, contains x_j.</td></tr>
<tr><td>in,out</td><td>u</td><td>is a one-dimensional array of size m and type double, that should be used and reset appropriately when status = 1 to 5 as directed by status.</td></tr>
<tr><td>in,out</td><td>v</td><td>is a one-dimensional array of size n and type double, that should be used and reset appropriately when status = 1 to 5 as directed by status.</td></tr>
</table>

Examples

lstrt.c.

3.1.2.5 lstr_information()

```
void lstr_information (
            void ** data,
            struct lstr_inform_type * inform,
            int * status )
```

Provides output information

Parameters

in,out	*data*	holds private internal data
out	*inform*	is a struct containing output information (see lstr_inform_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The values were recorded succesfully

Examples

lstrt.c.

3.1.2.6 lstr_terminate()

```
void lstr_terminate (
            void ** data,
            struct lstr_control_type * control,
            struct lstr_inform_type * inform )
```

Deallocate all internal private storage

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see lstr_control_type)
out	*inform*	is a struct containing output information (see lstr_inform_type)

Examples

lstrt.c.

Chapter 4

Example Documentation

4.1 lstrt.c

This is an example of how to use the package to solve a trust-region problem. The use of default and non-default scaling matrices, and restarts with a smaller trust-region radius are illustrated.

```
/* lstrt.c */
/* Full test for the LSTR C interface */
#include <stdio.h>
#include <math.h>
#include "lstr.h"
int main(void) {
    // Derived types
    void *data;
    struct lstr_control_type control;
    struct lstr_inform_type inform;
    // Set problem data
    int n = 50; // dimensions
    int m = 2 * n;
    int status;
    double radius;
    double x[n];
    double u[m];
    double v[n];
    // Initialize lstr
    lstr_initialize( &data, &control, &status );
    // resolve with a smaller radius ?
    for( int new_radius=0; new_radius <= 1; new_radius++){
      if ( new_radius == 0 ){ // original radius
         radius = 1.0;
         status = 1;
      } else { // smaller radius
         radius = 0.1;
         status = 5;
      }
      control.print_level = 0;
      lstr_import_control( &control, &data, &status );
      for( int i = 0; i < m; i++) u[i] = 1.0; // b = 1
      // iteration loop to find the minimizer with A^T = (I:diag(1:n))
      while(true){ // reverse-communication loop
        lstr_solve_problem( &data, &status, m, n, radius, x, u, v );
        if ( status == 0 ) { // successful termination
            break;
        } else if ( status < 0 ) { // error exit
            break;
        } else if ( status == 2 ) { // form u <- u + A * v
          for( int i = 0; i < n; i++) {
            u[i] = u[i] + v[i];
            u[n+i] = u[n+i] + (i+1)*v[i];
          }
        } else if ( status == 3 ) { // form v <- v + A^T * u
          for( int i = 0; i < n; i++) v[i] = v[i] + u[i] + (i+1) * u[n+i];
        } else if ( status == 4 ) { // restart
          for( int i = 0; i < m; i++) u[i] = 1.0;
        }else{
            printf(" the value %1i of status should not occur\n",
```

```
              status);
            break;
          }
        }
        lstr_information( &data, &inform, &status );
        printf("%1i lstr_solve_problem exit status = %i,"
             " f = %.2f\n", new_radius, inform.status, inform.r_norm );
      }
   // Delete internal workspace
   lstr_terminate( &data, &control, &inform );
}
```

Index