C interfaces to GALAHAD GLTR

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory

Sun Apr 16 2023

Chapter 1

GALAHAD C package gltr

1.1 Introduction

1.1.1 Purpose

Given real n by n symmetric matrices H and M (with M positive definite), a real n vector c and scalars $\Delta > 0$ and f_0, this package finds an **approximate minimizer of the quadratic objective function $\frac{1}{2}x^T Hx + c^T x + f_0$, where the vector x is required to satisfy the constraint $\|x\|_M \leq \Delta$**, and where the M-norm of x is $\|x\|_M = \sqrt{x^T Mx}$. This problem commonly occurs as a trust-region subproblem in nonlinear optimization calculations. The method may be suitable for large n as no factorization of H is required. Reverse communication is used to obtain matrix-vector products of the form Hz and $M^{-1}z$.

The package may also be used to solve the related problem in which x is instead required to satisfy the **equality constraint** $\|x\|_M = \Delta$.

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

April 1997, C interface December 2021.

1.1.4 Terminology

1.1.5 Method

The required solution x necessarily satisfies the optimality condition $Hx + \lambda Mx + c = 0$, where $\lambda \geq 0$ is a Lagrange multiplier corresponding to the constraint $\|x\|_M \leq \Delta$. In addition, the matrix $H + \lambda M$ will be positive definite.

The method is iterative. Starting with the vector $M^{-1}c$, a matrix of Lanczos vectors is built one column at a time so that the k-th column is generated during iteration k. These columns span a so-called Krylov space. The resulting n by k matrix Q_k has the property that $Q_k^T H Q_k = T_k$, where T_k is tridiagonal. An approximation to the required solution may then be expressed formally as

$$x_{k+1} = Q_k y_k,$$

where y_k solves the \`'tridiagonal'' subproblem of minimizing

$$(1) \quad \frac{1}{2} y^T T_k y + \|c\|_{M^{-1}} e_1^T y \text{ subject to the constraint } \|y\|_2 \leq \Delta,$$

and where e_1 is the first unit vector.

If the solution to (1) lies interior to the constraint, the required solution x_{k+1} may simply be found as the k-th (preconditioned) conjugate-gradient iterate. This solution can be obtained without the need to access the whole matrix Q_k. These conjugate-gradient iterates increase in M-norm, and thus once one of them exceeds Δ in M-norm, the solution must occur on the constraint boundary. Thereafter, the solution to (1) is less easy to obtain, but an efficient inner iteration to solve (1) is nonetheless achievable because T_k is tridiagonal. It is possible to observe the optimality measure $\|Hx + \lambda Mx + c\|_{M^{-1}}$ without computing x_{k+1}, and thus without needing Q_k. Once this measure is sufficiently small, a second pass is required to obtain the estimate x_{k+1} from y_k. 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.

A cheaper alternative is to use the Steihuag-Toint strategy, which is simply to stop at the first boundary point encountered along the piecewise linear path generated by the conjugate-gradient iterates. Note that if H is significantly indefinite, this strategy often produces a far from optimal point, but is effective when H is positive definite or almost

1.1.6 Reference

The method is described in detail in

N. I. M. Gould, S. Lucidi, M. Roma and Ph. L. Toint, Solving the trust-region subproblem using the Lanczos method. SIAM Journal on Optimization **9:2** (1999), 504-525.

1.1.7 Call order

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

- gltr_initialize - provide default control parameters and set up initial data structures
- gltr_read_specfile (optional) - override control values by reading replacement values from a file
- gltr_import_control - import control parameters prior to solution
- gltr_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
- gltr_information (optional) - recover information about the solution and solution process
- gltr_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 galahad_gltr.h File Reference

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

Data Structures

- struct gltr_control_type
- struct gltr_inform_type

Functions

- void gltr_initialize (void **data, struct gltr_control_type *control, int *status)
- void gltr_read_specfile (struct gltr_control_type *control, const char specfile[ ])
- void gltr_import_control (struct gltr_control_type *control, void **data, int *status)
- void gltr_solve_problem (void **data, int *status, int n, const real_wp_ radius, real_wp_ x[ ], real_wp_ r[ ], real_wp_ vector[ ])
- void gltr_information (void **data, struct gltr_inform_type *inform, int *status)
- void gltr_terminate (void **data, struct gltr_control_type *control, struct gltr_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct gltr_control_type

control derived type as a C struct

Examples

gltrt.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	itmax	the maximum number of iterations allowed (-ve = no bound)
int	Lanczos_itmax	the maximum number of iterations allowed once the boundary has been encountered (-ve = no bound)
int	extra_vectors	the number of extra work vectors of length n used
int	ritz_printout_device	the unit number for writing debug Ritz values
real_wp_	stop_relative	the iteration stops successfully when the gradient in the M(inverse) nor is smaller than max(stop_relative $*$ initial M(inverse) gradient norm, 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_	f_min	the iteration stops if the objective-function value is lower than f_min
real_wp_	rminvr_zero	the smallest value that the square of the M norm of the gradient of the the objective may be before it is considered to be zero
real_wp_	f_0	the constant term, f_0, in the objective function
bool	unitm	is M the identity matrix ?
bool	steihaug_toint	should the iteration stop when the Trust-region is first encountered ?
bool	boundary	is the solution thought to lie on the constraint boundary ?
bool	equality_problem	is the solution required to lie on the constraint boundary ?
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
bool	print_ritz_values	should the Ritz values be written to the debug stream?
char	ritz_file_name[31]	name of debug file containing the Ritz values
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 gltr_inform_type

inform derived type as a C struct

Examples

gltrt.c.

Data Fields

int	status	return status. See gltr_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

Data Fields

int	iter_pass2	the total number of pass-2 iterations required if the solution lies on the trust-region boundary
real_wp_	obj	the value of the quadratic function
real_wp_	multiplier	the Lagrange multiplier corresponding to the trust-region constraint
real_wp_	mnormx	the M-norm of x
real_wp_	piv	the latest pivot in the Cholesky factorization of the Lanczos tridiagona
real_wp_	curv	the most negative cuurvature encountered
real_wp_	rayleigh	the current Rayleigh quotient
real_wp_	leftmost	an estimate of the leftmost generalized eigenvalue of the pencil (H,M)
bool	negative_curvature	was negative curvature encountered ?
bool	hard_case	did the hard case occur ?

3.1.2 Function Documentation

3.1.2.1 gltr_initialize()

```
void gltr_initialize (
            void ** data,
            struct gltr_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 gltr_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

gltrt.c.

3.1.2.2 gltr_read_specfile()

```
void gltr_read_specfile (
            struct gltr_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 RUNGLTR.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in $GALAHAD/doc/gltr.pdf for a list of keywords that may be set.

Parameters

in,out	*control*	is a struct containing control information (see gltr_control_type)
in	*specfile*	is a character string containing the name of the specification file

3.1.2.3 gltr_import_control()

```
void gltr_import_control (
            struct gltr_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 gltr_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

gltrt.c.

3.1.2.4 gltr_solve_problem()

```
void gltr_solve_problem (
            void ** data,
            int * status,
            int n,
            const real_wp_ radius,
            real_wp_ x[ ],
            real_wp_ r[ ],
            real_wp_ vector[ ] )
```

Solve the trust-region problem using reverse communication.

Parameters

in,out	*data*	holds private internal data
in,out	*status*	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 r (below) to c for this entry. • 4. the iteration is to be restarted with a smaller radius but with all other data unchanged. Set r (below) to c for this entry. Possible exit values are: • 0. the solution has been found • 2. the inverse of M must be applied to vector with the result returned in vector and the function re-entered with all other data unchanged. This will only happen if control.unitm is false • 3. the product $H *$ vector must be formed, with the result returned in vector and the function re-entered with all other data unchanged • 5. The iteration must be restarted. Reset r (below) to c and re-enter with all other data unchanged. This exit will only occur if control.steihaug_toint is false and the solution lies on the trust-region boundary • -1. an array allocation has failed • -2. an array deallocation has failed • -3. n and/or radius is not positive • -15. the matrix M appears to be indefinite • -18. the iteration limit has been exceeded • -30. the trust-region has been encountered in Steihaug-Toint mode • -31. the function value is smaller than control.f_min
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*radius*	is a scalar of type double, that holds the trust-region radius, Δ, used. radius must be strictly positive
in,out	*x*	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.
in,out	*r*	is a one-dimensional array of size n and type double, that that must be set to c on entry (status = 1) and re-entry ! (status = 4, 5). On exit, r contains the resiual $Hx + c$.
in,out	*vector*	is a one-dimensional array of size n and type double, that should be used and reset appropriately when status = 2 and 3 as directed.

Examples

gltrt.c.

3.1.2.5 gltr_information()

```
void gltr_information (
            void ** data,
```

```
            struct gltr_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 gltr_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

gltrt.c.

3.1.2.6 gltr_terminate()

```
void gltr_terminate (
            void ** data,
            struct gltr_control_type * control,
            struct gltr_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 gltr_control_type)
out	*inform*	is a struct containing output information (see gltr_inform_type)

Examples

gltrt.c.

Chapter 4

Example Documentation

4.1 gltrt.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.

```
/* gltrt.c */
/* Full test for the GLTR C interface */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_gltr.h"
int main(void) {
    // Derived types
    void *data;
    struct gltr_control_type control;
    struct gltr_inform_type inform;
    // Set problem data
    int n = 100; // dimension
    int status;
    real_wp_ radius;
    real_wp_ x[n];
    real_wp_ r[n];
    real_wp_ vector[n];
    real_wp_ h_vector[n];
    // Initialize gltr
    gltr_initialize( &data, &control, &status );
    // use a unit M ?
    for( int unit_m=0; unit_m <= 1; unit_m++){
      if ( unit_m == 0 ){
        control.unitm = false;
      } else {
        control.unitm = true;
      }
      gltr_import_control( &control, &data, &status );
      // resolve with a smaller radius ?
      for( int new_radius=0; new_radius <= 1; new_radius++){
        if ( new_radius == 0 ){
           radius = 1.0;
           status = 1;
        } else {
           radius = 0.1;
           status = 4;
        }
        for( int i = 0; i < n; i++) r[i] = 1.0;
        // iteration loop to find the minimizer
        while(true){ // reverse-communication loop
          gltr_solve_problem( &data, &status, n, radius, x, r, vector );
          if ( status == 0 ) { // successful termination
              break;
          } else if ( status < 0 ) { // error exit
              break;
          } else if ( status == 2 ) { // form the preconditioned vector
            for( int i = 0; i < n; i++) vector[i] = vector[i] / 2.0;
          } else if ( status == 3 ) { // form the Hessian-vector product
```

```
            h_vector[0] =  2.0 * vector[0] + vector[1];
            for( int i = 1; i < n-1; i++){
              h_vector[i] = vector[i-1] + 2.0 * vector[i] + vector[i+1];
            }
            h_vector[n-1] = vector[n-2] + 2.0 * vector[n-1];
            for( int i = 0; i < n; i++) vector[i] = h_vector[i];
          } else if ( status == 5 ) { // restart
            for( int i = 0; i < n; i++) r[i] = 1.0;
          }else{
              printf(" the value %1i of status should not occur\n",
                status);
              break;
          }
        }
        gltr_information( &data, &inform, &status );
        printf("MR = %1i%1i gltr_solve_problem exit status = %i,"
             " f = %.2f\n", unit_m, new_radius, inform.status, inform.obj );
      }
    }
   // Delete internal workspace
   gltr_terminate( &data, &control, &inform );
}
```