C interfaces to GALAHAD GLRT

Jari Fowkes and Nick Gould

STFC Rutherford Appleton Laboratory

Sat Mar 26 2022

Chapter 1

GALAHAD C package glrt

1.1 Introduction

1.1.1 Purpose

Given real n by n symmetric matrices H and M (with M positive definite), real n vector c, and scalars $\sigma \geq 0$ and f_0, this package finds an **approximate minimizer of the regularised quadratic objective function**

$$\frac{1}{2}x^T Hx + c^T x + f_0 + \frac{1}{p}\sigma\|x\|_M^p,$$

where $\|v\|_M = \sqrt{v^T Mv}$ is the M-norm of v. This problem commonly occurs as a subproblem in nonlinear optimization calculations involving cubic regularisation. 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 $M^{-1}z$.

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

1.1.5 Method

The required solution x necessarily satisfies the optimality condition $Hx + \lambda Mx + c + \lambda o = 0$, where $\lambda = \sigma[\|x\|_M^2]^{p/2-1}$. In addition, the matrix $H + \lambda M$ will be positive semi-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

$$(1) \quad \frac{1}{2} y^T T_k y + \|c\|_{M^{-1}} e_1^T y + \frac{1}{p}\sigma \|y\|_2^p,$$

where e_1 is the first unit vector.

To minimize (1), the optimality conditions

$$(2) \quad (T_k + \lambda I) y(\lambda) = -c - \lambda d,$$

where $\lambda = \sigma \|y(\lambda) + d\|_M^{p-2}$ are used as the basis of an iteration. Specifically, given an estimate λ for which $T_k + \lambda I$ is positive definite, the tridiagonal system (2) may be efficiently solved to give $y(\lambda)$. It is then simply a matter of adjusting λ (for example by a Newton-like process) to solve the scalar nonlinear equation

$$(3) \quad \theta(\lambda) \equiv \|y(\lambda) + d\|_M^{p-2} - \frac{\lambda}{\sigma} = 0.$$

In practice (3) is reformulated, and a more rapidly converging iteration is used.

It is possible to measure the optimality measure $\|Hx + \lambda Mx + c + \lambda o\|_{M^{-1}}$ 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.

Special code is used in the special case $p = 2$, as in this case a single pass suffices.

1.1.6 Reference

The method is described in detail in

C. Cartis, N. I. M. Gould and Ph. L. Toint, Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results. Mathematical Programming **127(2)**, pp.245-295, 2011.

1.1.7 Call order

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

- glrt_initialize - provide default control parameters and set up initial data structures
- glrt_read_specfile (optional) - override control values by reading replacement values from a file
- glrt_import_control - import control parameters prior to solution
- glrt_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
- glrt_information (optional) - recover information about the solution and solution process
- glrt_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_glrt.h File Reference

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

Data Structures

- struct glrt_control_type
- struct glrt_inform_type

Functions

- void glrt_initialize (void **data, struct glrt_control_type *control, int *status)
- void glrt_read_specfile (struct glrt_control_type *control, const char specfile[ ])
- void glrt_import_control (struct glrt_control_type *control, void **data, int *status)
- void glrt_solve_problem (void **data, int *status, int n, const real_wp_ power, const real_wp_ weight, real←_wp_ x[ ], real_wp_ r[ ], real_wp_ vector[ ])
- void glrt_information (void **data, struct glrt_inform_type *inform, int *status)
- void glrt_terminate (void **data, struct glrt_control_type *control, struct glrt_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct glrt_control_type

control derived type as a C struct

Examples

glrtt.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	stopping_rule	the stopping rule used (see below): • 1. stop rule = norm of the step • 2. stop rule is norm of the step / σ other. stop rule = 1.0,
int	freq	frequency for solving the reduced tri-diagonal problem
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^{-1} norm is smaller than max(stop_relative $*$ min(1, stop_rule) $*$ norm initial gradient, 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_	rminvr_zero	the smallest value that the square of the M norm of the gradient of the objective may be before it is considered to be zero
real_wp_	f_0	the constant term, f0, in the objective function
bool	unitm	is M the identity matrix ?
bool	impose_descent	is descent required i.e., should $c^T x < 0$?
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 glrt_inform_type

inform derived type as a C struct

Examples

glrtt.c.

Data Fields

int	status	return status. See glrt_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

Data Fields

int	iter	the total number of iterations required
int	iter_pass2	the total number of pass-2 iterations required
real_wp_	obj	the value of the quadratic function
real_wp_	obj_regularized	the value of the regularized quadratic function
real_wp_	multiplier	the multiplier, $\sigma \|x\|^{p-2}$
real_wp_	xpo_norm	the value of the norm $\|x\|_M$
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 glrt_initialize()

```
void glrt_initialize (
            void ** data,
            struct glrt_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 glrt_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

glrtt.c.

3.1.2.2 glrt_read_specfile()

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

Parameters

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

3.1.2.3 glrt_import_control()

```
void glrt_import_control (
            struct glrt_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 glrt_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

glrtt.c.

3.1.2.4 glrt_solve_problem()

```
void glrt_solve_problem (
            void ** data,
            int * status,
            int n,
            const real_wp_ power,
            const real_wp_ weight,
            real_wp_ x[ ],
            real_wp_ r[ ],
            real_wp_ vector[ ] )
```

Solve the regularized-quadratic problem using reverse communication.

Parameters

in,out	*data*	holds private internal data

Parameters

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. • 6. the iteration is to be restarted with a larger weight 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 • 4. The iteration must be restarted. Reset r (below) to c and re-enter with all other data unchanged. • -1. an array allocation has failed • -2. an array deallocation has failed • -3. n and/or radius is not positive • -7. the problem is unbounded from below. This can only happen if power = 2, and in this case the objective is unbounded along the arc x + t vector as t goes to infinity • -15. the matrix M appears to be indefinite • -18. the iteration limit has been exceeded
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*power*	is a scalar of type double, that holds the egularization power, $p \geq 2$
in	*weight*	is a scalar of type double, that holds the positive regularization weight, σ
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

glrtt.c.

3.1.2.5 glrt_information()

```
void glrt_information (
            void ** data,
```

```
struct glrt_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 glrt_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

glrtt.c.

3.1.2.6 glrt_terminate()

```
void glrt_terminate (
            void ** data,
            struct glrt_control_type * control,
            struct glrt_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 glrt_control_type)
out	*inform*	is a struct containing output information (see glrt_inform_type)

Examples

glrtt.c.

Chapter 4

Example Documentation

4.1 glrtt.c

This is an example of how to use the package to solve a regularized quadratic problem. The use of default and non-default scaling matrices, and restarts with a larger regularization weight are illustrated.

```
/* glrtt.c */
/* Full test for the GLRT C interface */
#include <stdio.h>
#include <math.h>
#include "galahad_glrt.h"
int main(void) {
    // Derived types
    void *data;
    struct glrt_control_type control;
    struct glrt_inform_type inform;
    // Set problem data
    int n = 100; // dimension
    int status;
    double weight;
    double power = 3.0;
    double x[n];
    double r[n];
    double vector[n];
    double h_vector[n];
    // Initialize glrt
    glrt_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;
      }
      glrt_import_control( &control, &data, &status );
      // resolve with a larger weight ?
      for( int new_weight=0; new_weight <= 1; new_weight++){
        if ( new_weight == 0 ){
           weight = 1.0;
           status = 1;
        } else {
           weight = 10.0;
           status = 6;
        }
        for( int i = 0; i < n; i++) r[i] = 1.0;
        // iteration loop to find the minimizer
        while(true){ // reverse-communication loop
          glrt_solve_problem( &data, &status, n, power, weight, 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 == 4 ) { // 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;
          }
        }
        glrt_information( &data, &inform, &status );
        printf("MR = %1i%1i glrt_solve_problem exit status = %i,"
             " f = %.2f\n", unit_m, new_weight, inform.status,
                           inform.obj_regularized );
      }
    }
  // Delete internal workspace
  glrt_terminate( &data, &control, &inform );
}
```

Index