GALAHAD GLTR

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

1 SUMMARY

Given real n by n symmetric matrices $\mathbf{H}$ and $\mathbf{M}$ (with $\mathbf{M}$ positive definite), a real n vector $\mathbf{c}$ and scalars $\Delta > 0$ and f_0, this package finds an **approximate minimizer of the quadratic objective function** $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} + f_0$**, where the vector x is required to satisfy the constraint** $\|\mathbf{x}\|_{\mathbf{M}} \leq \Delta$, and where the $\mathbf{M}$-norm of $\mathbf{x}$ is $\|\mathbf{x}\|_{\mathbf{M}} = \sqrt{\mathbf{x}^T\mathbf{M}\mathbf{x}}$. 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 $\mathbf{H}$ is required. Reverse communication is used to obtain matrix-vector products of the form $\mathbf{H}\mathbf{z}$ and $\mathbf{M}^{-1}\mathbf{z}$.

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

ATTRIBUTES — Versions: `GALAHAD_GLTR_single`, `GALAHAD_GLTR_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_RAND`, `GALAHAD_NORMS`, `GALAHAD_ROOTS`, `GALAHAD_SPECFILE`, `*TTRF`. **Date:** April 1997. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

The package is available using both single and double precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_GLTR_single
```

with the obvious substitution `GALAHAD_GLTR_double`, `GALAHAD_GLTR_single_64` and `GALAHAD_GLTR_double_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `GLTR_control_type`, `GLTR_inform_type`, `GLTR_data_type`, (Section 2.2) and the subroutines `GLTR_initialize`, `GLTR_solve`, `GLTR_terminate` (Section 2.3) and `GLTR_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions and `DOUBLE PRECISION` for the double precision cases, and correspond to `rp_ = real32` and `rp_ = real64`, respectively, as supplied by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.2 The derived data types

Three derived data types are accessible from the package.

2.2.1 The derived data type for holding control parameters

The derived data type GLTR_control_type is used to hold controlling data. Default values may be obtained by calling GLTR_initialize (see Section 2.3.1). The components of GLTR_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in GLTR_solve and GLTR_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in GLTR_solve is suppressed if out < 0. The default is out = - 1.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level = 1 a single line of output will be produced for each iteration of the process. If print_level ≥ 2 this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

itmax is a scalar variable of type INTEGER(ip_), that holds the maximum number of iterations which will be allowed in GLTR_solve. If itmax is set to a negative number, it will be reset by GLTR_solve to n. The default is itmax = -1.

lanczos_itmax is a scalar variable of type INTEGER(ip_), that holds the maximum number of iterations that may be performed when the iterates encounter the boundary of the constraint. If lanczos_itmax is set to a negative number, it will be reset by GLTR_solve to n. The default is lanczos_itmax = -1.

unitm is a scalar variable of type default LOGICAL, that must be set .TRUE. if the matrix $\mathbf{M}$ is the identity matrix, and .FALSE. otherwise. The default is unitm = .TRUE..

extra_vectors is a scalar variable of type INTEGER(ip_), that specifies the number of additional vectors of length n that will be allocated to try to speed up the computation if the constraint boundary is encountered. The default is extra_vectors = 0.

steihaug_toint is a scalar variable of type default LOGICAL, which must be set .TRUE. if the algorithm is required to stop at the first point on the boundary of the constraint that is encountered, and .FALSE. if the constraint boundary is to be investigated further. Setting steihaug_toint to .TRUE. can reduce the amount of computation at the expense of obtaining a poorer estimate of the solution. The default is steihaug_toint = .FALSE..

boundary is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user believes that the solution will occur on the constraint boundary, and .FALSE. otherwise. A correct setting of boundary may reduce the amount of computation performed. The default is boundary = .FALSE..

equality_problem is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user requires that the solution occur on the constraint boundary (i.e., that the inequality constraint be replaced by $\|\mathbf{x}\|_{\mathbf{M}} = \Delta$), and .FALSE. otherwise. The default is equality_problem = .FALSE..

space_critical is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to allocate as little internal storage as possible, and .FALSE. otherwise. The package may be more efficient if space_critical is set .FALSE.. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to return to the user in the unlikely event that an internal array deallocation fails, and .FALSE. if the package should be allowed to try to continue. The default is deallocate_error_fatal = .FALSE..

stop_relative and stop_absolute are scalar variables of type REAL(rp_), that holds the relative and absolute convergence tolerances (see Section 4). The computed solution $\mathbf{x}$ is accepted by GLTR_solve if the computed value of $\|\mathbf{H}\mathbf{x} + \lambda\mathbf{M}\mathbf{x} + \mathbf{c}\|_{\mathbf{M}^{-1}}$ is less than or equal to max($\|\mathbf{c}\|_{\mathbf{M}^{-1}} *$ stop_relative, stop_absolute), where λ is an estimate of the Lagrange multiplier associated with the trust-region constraint. The defaults are stop_relative = $\sqrt{u}$ and stop_absolute = 0.0, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_GLTR_double).

f_min is a scalar variable of type default REAL(rp_), that specifies the smallest value of the objective function that is allowed without regarding the problem as being unbounded from below. The default is f_min = -HUGE(1.0)/2.0 (-HUGE(1.0D0)/2.0D0 in GALAHAD_GLTR_double)

fraction_opt is a scalar variable of type default REAL(rp_), that specifies the fraction of the optimal value which is to be considered acceptable by the algorithm. A negative value is considered to be zero, and a value of larger than one is considered to be one. Reducing fraction_opt below one will result in a reduction of the computation performed at the expense of an inferior optimal value. The default is fraction_opt = 1.0.

f_0 is a scalar variable of type default REAL(rp_), that gives the value of the constant term f_0 in the quadratic objective function. This value has no effect on the computed minimizer $\mathbf{x}$. The default is f_0 = 0.0.

rminvr_zero is a scalar variable of type default REAL(rp_), that gives the smallest value that the square of the $\mathbf{M}$-norm of the gradient of the objective function may be before it is considered to be zero. The default is rminvr_zero = 10 u, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_GLTR_double).

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

2.2.2 The derived data type for holding informational parameters

The derived data type GLTR_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of GLTR_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the current status of the algorithm. See Sections 2.4 and 2.5 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last internal array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

multiplier is a scalar variable of type default REAL(rp_), that holds the value of the Lagrange multiplier associated with the constraint.

mnormx is a scalar variable of type default REAL(rp_), that holds the current value of $\|\mathbf{x}\|_{\mathbf{M}}$.

leftmost is a scalar variable of type default REAL(rp_), that holds an estimate of the leftmost eigenvalue of the matrix pencil $(\mathbf{H},\mathbf{M})$.

iter is a scalar variable of type INTEGER(ip_), that holds the current number of Lanczos vectors used.

iter_pass2 is a scalar variable of type INTEGER(ip_), that holds the current number of Lanczos vectors used in the second pass.

negative_curvature is a scalar variable of type default LOGICAL, that is set .TRUE. if $\mathbf{H}$ has been found to be indefinite during the calculation and .FALSE. otherwise.

For backward compatibility with an earlier version of the package, there is an equivalent type GLTR_info_type, and the two names may be used interchangeably.

2.2.3 The derived data type for holding problem data

The derived data type GLTR_data_type is used to hold all the data for a particular problem between calls of GLTR procedures. This data should be preserved, untouched, from the initial call to GLTR_initialize to the final call to GLTR_terminate.

2.3 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.7 for further features):

1. The subroutine GLTR_initialize is used to set default values, and initialize private data.
2. The subroutine GLTR_solve is called repeatedly to solve the problem. On each exit, the user may be expected to provide additional information and, if necessary, re-enter the subroutine.
3. The subroutine GLTR_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by GLTR_solve, at the end of the solution process. It is important to do this if the data object is re-used for another problem since GLTR_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We use square brackets [ ] to indicate OPTIONAL arguments.

2.3.1 The initialization subroutine

Default values are provided as follows:

```
CALL GLTR_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type GLTR_data_type (see Section 2.2.3). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type GLTR_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling GLTR_initialize.

inform is a scalar INTENT(OUT) argument of type GLTR_inform_type (see Section 2.2.2). A successful call to GLTR_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.5.

2.3.2 The optimization problem solution subroutine

The optimization problem solution algorithm is called as follows:

```
CALL GLTR_solve( n, radius, f, X, R, VECTOR, data, control, inform )
```

n is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the number of unknowns, n. **Restriction:** $n > 0$.

radius is a scalar INTENT(IN) variable of type default REAL(rp_), that must be set on initial entry to the value of the radius of the quadratic constraint, Δ. **Restriction:** $\Delta > 0$.

f is a scalar INTENT(INOUT) variable of type default REAL(rp_), that holds an estimate of the optimal value of the quadratic objective function. On initial entry, f need not be set. It must not be changed between entries. On exit, f contains the current best estimate of the optimal objective value.

X is an array INTENT(INOUT) argument of dimension n and type REAL(rp_), that holds an estimate of the solution $\mathbf{x}$ of the linear system. On initial entry, X need not be set. It must not be changed between entries. On exit, X contains the current best estimate of the solution.

R is an array INTENT(INOUT) argument of dimension n and type REAL(rp_), that is used to hold the gradient $\mathbf{H}\mathbf{x}+\mathbf{c}$ of the objective function at the current estimate of the solution. On initial entry, R must contain the vector $\mathbf{c}$. If inform%status = 5 on exit, it must be reset to $\mathbf{c}$; otherwise it must be left unchanged. On exit, R contains the gradient of the objective function at the current best estimate of the solution.

VECTOR is an array INTENT(INOUT) argument of dimension n and type REAL(rp_), that is used to pass information from and to GLTR_solve, as explained in Section 2.4. On initial entry, VECTOR need not be set. On exit, the actual content of the array depends on the value of the parameter inform%status (see Section 2.4).

data is a scalar INTENT(INOUT) argument of type GLTR_data_type (see Section 2.2.3). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to GLTR_initialize.

control is a scalar INTENT(IN) argument of type GLTR_control_type. (see Section 2.2.1). Default values may be assigned by calling GLTR_initialize prior to the first call to GLTR_solve.

inform is a scalar INTENT(INOUT) argument of type GLTR_inform_type (see Section 2.2.2). On initial entry, the component status must be set to 1. The remaining components need not be set. A successful call to GLTR_solve is indicated when the component status has the value 0. For other return values of status, see Sections 2.4 and 2.5.

2.3.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL GLTR_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type GLTR_data_type exactly as for GLTR_solve that must not have been altered **by the user** since the last call to GLTR_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type GLTR_control_type exactly as for GLTR_solve.

inform is a scalar INTENT(OUT) argument of type GLTR_inform_type exactly as for GLTR_solve. Only the component status will be set on exit, and a successful call to GLTR_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.5.

2.4 Reverse communication

A positive value of inform%status on exit from GLTR_solve indicates that the user needs to take appropriate action before re-entering the subroutine. Possible values are:

2. The user must perform the preconditioning operation

$$\mathbf{y} := \mathbf{M}^{-1}\mathbf{z},$$

and recall GLTR_solve. The vector $\mathbf{z}$ is available in the array VECTOR, and the result $\mathbf{y}$ must be placed in VECTOR. No argument except VECTOR should be altered before recalling GLTR_solve. This return can only occur when control%unitm is .FALSE..

3. The user must perform the matrix-vector product

$$\mathbf{y} := \mathbf{H}\mathbf{z}$$

and recall GLTR_solve. The vector $\mathbf{z}$ is available in the array VECTOR, and the result $\mathbf{y}$ must be placed in VECTOR. No argument except VECTOR should be altered before recalling GLTR_solve.

5. The user should reset R to $\mathbf{c}$ and recall GLTR_solve. No argument except R should be altered before recalling GLTR_solve.

2.5 Warning and error messages

A negative value of inform%status on exit from GLTR_solve or GLTR_terminate indicates that an error might have occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc respectively.

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc respectively.

-3. (GLTR_solve only) One of the restrictions n > 0 or radius > 0 has been violated.

-15. (GLTR_solve only) The matrix $\mathbf{M}$ appears not to be positive definite.

-18. (GLTR_solve only) More than control%itmax iterations have been performed without obtaining convergence.

-30. (GLTR_solve only) The constraint boundary has been encountered when the input value of control%steihaug_toint was set .TRUE.. The solution is unlikely to have achieved the accuracy required by control%stop_relative and control%stop_absolute.

-44 (GLTR_solve only) An objective function value smaller than control%f_min has been detected.

2.6 Re-entry with a new value of Δ

It commonly happens that, having solved the problem for a particular value of the radius Δ, a user now wishes to solve the problem for a different value of Δ. Rather than restarting the calculation with inform%status = 1, a useful approximation may be found resetting radius to the new required value and R to $\mathbf{c}$, and recalling GLTR_solve with inform%status = 4 and the remaining arguments unchanged. This will determine the best solution within the Krylov space investigated in the previous minimization (see Section 4).

2.7 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type GLTR_control_type (see Section 2.2.1), by reading an appropriate data specification file using the subroutine GLTR_read_specfile. This facility is useful as it allows a user to change GLTR control parameters without editing and recompiling programs that call GLTR.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more

than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by GLTR_read_specfile must start with a "BEGIN GLTR" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by GLTR_read_specfile .. )
  BEGIN GLTR
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by GLTR_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN GLTR" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN GLTR SPECIFICATION
```

and

```
END GLTR SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN GLTR" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when GLTR_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by GLTR_read_specfile.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL GLTR_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type GLTR_control_type (see Section 2.2.1). Default values should have already been set, perhaps by calling GLTR_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.2.1) of control that each affects are given in Table 2.1.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
maximum-number-of-iterations	%itmax	integer
maximum-number-of-Lanczos-iterations	%lanczos_itmax	integer
number-extra-n-vectors-used	%extra_vectors	integer
relative-accuracy-required	%stop_relative	real
absolute-accuracy-required	%stop_absolute	real
small-f-stop	%f_min	real
fraction-optimality-required	%fraction_opt	real
constant-term-in-objective	%f_0	real
zero-gradient-tolerance	%rminvr_zero	real
two-norm-trust-region	%unitm	logical
stop-as-soon-as-boundary-encountered	%steihaug_toint	logical
solution-is-likely-on-boundary	%boundary	logical
equality-problem	%equality_problem	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical

Table 2.1: Specfile commands and associated components of control.

2.8 Information printed

If control%print_level is positive, information about the progress of the algorithm will be printed on unit control%out. If control%print_level = 1, a single line of output will be produced for each iteration of the process. So long as the current estimate lies within the constraint boundary, this will include the iteration number, the value of the objective function, the norm of the gradient, the step taken, the two-norm of the latest search direction, the norm $\|\mathbf{x}\|_{\mathbf{M}}$, and the curvature of $\mathbf{H}$ observed along the current search direction (see Section 4). A further message will be printed if the constraint boundary is encountered during the current iteration. Thereafter, the one-line summary will record the iteration number, the value of the objective function, the norm of the gradient of the Lagrangian, $\|\mathbf{H}\mathbf{x}+\lambda\mathbf{M}\mathbf{x}+\mathbf{c}\|_{\mathbf{M}^{-1}}$, the value of the Lagrange multiplier, the number of iterations taken to solve the "tridiagonal" subproblem and the exit status from this calculation (once again, see Section 4). If control%print_level ≥ 2, this output will be increased to provide significant detail of each iteration. This extra output includes a complete history of the inner iteration required to solve the "tridiagonal" subproblem, and records the leftmost eigenvalue of $\mathbf{T}_k$ (if needed), along with the estimates of the Lagrange multipliers and the norm $\|\mathbf{x}\|_{\mathbf{M}}$ generated during the inner iteration.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: GLTR_solve calls the LAPACK subroutine *PTTRF, where * is S for the default real version and D for the double precision version.

Other modules used directly: GLTR_solve calls the GALAHAD packages GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD_RAND, GALAHAD_NORMS, GALAHAD_ROOTS and GALAHAD_SPECFILE.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: $n > 0$, $\Delta > 0$.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

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

The method is iterative. Starting with the vector $\mathbf{M}^{-1}\mathbf{c}$, 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 $\mathbf{Q}_k$ has the property that $\mathbf{Q}_k^T\mathbf{H}\mathbf{Q}_k = \mathbf{T}_k$, where $\mathbf{T}_k$ is tridiagonal. An approximation to the required solution may then be expressed formally as

$$\mathbf{x}_{k+1} = \mathbf{Q}_k\mathbf{y}_k$$

where $\mathbf{y}_k$ solves the "tridiagonal" subproblem of minimizing

$$\tfrac{1}{2}\mathbf{y}^T\mathbf{T}_k\mathbf{y} + \|\mathbf{c}\|_{\mathbf{M}^{-1}}\mathbf{e}_1^T\mathbf{y} \text{ subject to the constraint } \|\mathbf{y}\|_2 \leq \Delta, \tag{4.1}$$

and where $\mathbf{e}_1$ is the first unit vector.

If the solution to (4.1) lies interior to the constraint, the required solution $\mathbf{x}_{k+1}$ 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 $\mathbf{Q}_k$. These conjugate-gradient iterates increase in $\mathbf{M}$-norm, and thus once one of them exceeds Δ in $\mathbf{M}$-norm, the solution must occur on the constraint boundary. Thereafter, the solution to (4.1) is less easy to obtain, but an efficient inner iteration to solve (4.1) is nonetheless achievable because $\mathbf{T}_k$ is tridiagonal. It is possible to observe the optimality measure $\|\mathbf{H}\mathbf{x} + \lambda\mathbf{M}\mathbf{x} + \mathbf{c}\|_{\mathbf{M}^{-1}}$ without computing $\mathbf{x}_{k+1}$, and thus without needing $\mathbf{Q}_k$. Once this measure is sufficiently small, a second pass is required to obtain the estimate $\mathbf{x}_{k+1}$ from $\mathbf{y}_k$. 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 $\mathbf{H}$ is significantly indefinite, this strategy often produces a far from optimal point, but is effective when $\mathbf{H}$ is positive definite or almost so.

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.

5 EXAMPLE OF USE

Suppose we wish to solve a problem in 10,000 unknowns, whose data is

$$\mathbf{H} = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & . & & \\ & . & . & . & \\ & & . & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} 2 & & & & \\ & 2 & & & \\ & & . & & \\ & & & 2 & \\ & & & & 2 \end{pmatrix} \text{ and } \mathbf{c} = \begin{pmatrix} 1 \\ 1 \\ . \\ 1 \\ 1 \end{pmatrix},$$

with a radius $\Delta = 10$. Then we may use the following code

```
   PROGRAM GALAHAD_GLTR_EXAMPLE  !  GALAHAD 4.3 - 2022-11-26 AT 09:50 GMT.
   USE GALAHAD_GLTR_DOUBLE                       ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
   REAL ( KIND = wp ), PARAMETER :: one = 1.0_wp, two = 2.0_wp
   INTEGER, PARAMETER :: n = 10000               ! problem dimension
   INTEGER :: i
   REAL ( KIND = wp ) :: f, radius = 10.0_wp  ! radius of ten
   REAL ( KIND = wp ), DIMENSION( n ) :: X, R, VECTOR, H_vector
   TYPE ( GLTR_data_type ) :: data
   TYPE ( GLTR_control_type ) :: control
   TYPE ( GLTR_inform_type ) :: inform
   CALL GLTR_initialize( data, control, inform ) ! Initialize control parameters
!  control%print_level = 1
   control%unitm = .FALSE.                ! M is not the identity matrix
   R = one                                ! The linear term is a vector of ones
   inform%status = 1
   DO                                     !  Iteration to find the minimizer
     CALL GLTR_solve( n, radius, f, X, R, VECTOR, data, control, inform )
     SELECT CASE( inform%status )  ! Branch as a result of inform%status
     CASE( 2 )                  ! Form the preconditioned gradient
       VECTOR = VECTOR / two      ! Preconditioner is two times identity
     CASE ( 3 )                 ! Form the matrix-vector product
       H_vector( 1 ) = - two * VECTOR( 1 ) + VECTOR( 2 )
       DO i = 2, n - 1
         H_vector( i ) = VECTOR( i - 1 ) - two * VECTOR( i ) + VECTOR( i + 1 )
       END DO
       H_vector( n ) = VECTOR( n - 1 ) - two * VECTOR( n )
       VECTOR = H_vector
     CASE ( 5 )        !  Restart
       R = one
     CASE ( - 30, 0 )  !  Successful return
       WRITE( 6, "( I6, ' iterations. Solution and Lagrange multiplier = ',  &
      &    2ES12.4 )" ) inform%iter + inform%iter_pass2, f, inform%multiplier
       CALL GLTR_terminate( data, control, inform ) ! delete internal workspace
        EXIT
     CASE DEFAULT      !  Error returns
       WRITE( 6, "( ' GLTR_solve exit status = ', I6 ) " ) inform%status
       CALL GLTR_terminate( data, control, inform ) ! delete internal workspace
       EXIT
     END SELECT
   END DO
   END PROGRAM GALAHAD_GLTR_EXAMPLE
```

This produces the following output:

```
    11 iterations. Solution and Lagrange multiplier =  -7.0711E+02  7.0712E+00
```