


GALAHAD LSTR

USER DOCUMENTATION GALAHAD Optimization Library version 3.3

1 SUMMARY

Given a real m by n matrix $\mathbf{A}$, a real m vector $\mathbf{b}$ and a scalar $\Delta > 0$, this package finds an **approximate minimizer of $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$, where the vector $\mathbf{x}$ is required to satisfy the "trust-region" constraint $\|\mathbf{x}\|_2 \leq \Delta$.** 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 $\mathbf{A}$ is required. Reverse communication is used to obtain matrix-vector products of the form $\mathbf{u} + \mathbf{A}\mathbf{v}$ and $\mathbf{v} + \mathbf{A}^T\mathbf{u}$.

ATTRIBUTES — Versions: `GALAHAD_LSTR_single`, `GALAHAD_LSTR_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_NORMS`, `GALAHAD_ROOTS`, `GALAHAD_SPECFILE`, `*ROTG`. **Date:** November 2007. **Origin:** N. I. M. Gould, Oxford University and Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

Access to the package requires a `USE` statement such as

Single precision version

```
USE GALAHAD_LSTR_single
```

Double precision version

```
USE GALAHAD_LSTR_double
```

If it is required to use both modules at the same time, the derived types `LSTR_control_type`, `LSTR_inform_type`, `LSTR_data_type`, (Section 2.1) and the subroutines `LSTR_initialize`, `LSTR_solve`, `LSTR_terminate` (Section 2.2) and `LSTR_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 The derived data types

Three derived data types are accessible from the package.

2.1.1 The derived data type for holding control parameters

The derived data type `LSTR_control_type` is used to hold controlling data. Default values may be obtained by calling `LSTR_initialize` (see Section 2.2.1). The components of `LSTR_control_type` are:

`error` is a scalar variable of type default `INTEGER`, that holds the stream number for error messages. Printing of error messages in `LSTR_solve` and `LSTR_terminate` is suppressed if `error` $\leq$ `0`. The default is `error = 6`.

`out` is a scalar variable of type default `INTEGER`, that holds the stream number for informational messages. Printing of informational messages in `LSTR_solve` is suppressed if `out` $<$ `0`. The default is `out = 6`.

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

itmin is a scalar variable of type default INTEGER, that holds the minimum number of iterations which will be performed by LSTR_solve. The default is itmin = -1.

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

itmax_on_boundary is a scalar variable of type default INTEGER, that holds the maximum number of iterations that may be performed when the iterates encounter the boundary of the constraint. If itmax_on_boundary is set to a negative number, it will be reset by LSTR_solve to $\max(m,n)+1$. The default is itmax_on_boundary = -1.

bitmax is a scalar variable of type default INTEGER, that holds the maximum number of Newton inner iterations which will be allowed for each main iteration in LSTR_solve. If bitmax is set to a negative number, it will be reset by LSTR_solve to 10. The default is bitmax = -1.

extra_vectors is a scalar variable of type default INTEGER, 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 = .TRUE..

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 (double precision in GALAHAD_LSTR_double), that holds the relative and absolute convergence tolerances (see Section 4). The computed solution $\mathbf{x}$ is accepted by LSTR_solve if the computed value of $\|\mathbf{A}^T(\mathbf{A}\mathbf{x}-\mathbf{b})+\lambda\mathbf{x}\|_2$ is less than or equal to $\max(\|\mathbf{A}^T\mathbf{b}\|_2 *$ 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_LSTR_double).

fraction_opt is a scalar variable of type default REAL (double precision in GALAHAD_LSTR_double), 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.

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 = "".

2.1.2 The derived data type for holding informational parameters

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

status is a scalar variable of type default INTEGER, that gives the current status of the algorithm. See Sections 2.3 and 2.4 for details.

alloc_status is a scalar variable of type default INTEGER, 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 (double precision in GALAHAD_LSTR_double), that holds the value of the Lagrange multiplier λ associated with the constraint.

x_norm is a scalar variable of type default REAL (double precision in GALAHAD_LSTR_double), that holds the current value of $\|\mathbf{x}\|_2$.

r_norm is a scalar variable of type default REAL (double precision in GALAHAD_LSTR_double), that holds the current value of $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2$.

Atr_norm is a scalar variable of type default REAL (double precision in GALAHAD_LSTR_double), that holds the current value of $\|\mathbf{A}^T(\mathbf{A}\mathbf{x}-\boldsymbol{b})+\lambda\mathbf{x}\|_2$.

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

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

2.1.3 The derived data type for holding problem data

The derived data type LSTR_data_type is used to hold all the data for a particular problem between calls of LSTR procedures. This data should be preserved, untouched, from the initial call to LSTR_initialize to the final call to LSTR_terminate.

2.2 Argument lists and calling sequences

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

1. The subroutine LSTR_initialize is used to set default values, and initialize private data.
2. The subroutine LSTR_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 LSTR_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by LSTR_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 LSTR_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

2.2.1 The initialization subroutine

Default values are provided as follows:

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

data is a scalar INTENT(INOUT) argument of type LSTR_data_type (see Section 2.1.3). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type LSTR_control_type (see Section 2.1.1). On exit, control contains default values for the components as described in Section 2.1.1. These values should only be changed after calling LSTR_initialize.

inform is a scalar INTENT(OUT) argument of type LSTR_inform_type (see Section 2.1.2). A successful call to LSTR_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.2.2 The optimization problem solution subroutine

The optimization problem solution algorithm is called as follows:

```
CALL LSTR_solve( m, n, radius, X, U, V, data, control, inform )
```

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

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

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

X is an array INTENT(INOUT) argument of dimension n and type default REAL (double precision in GALAHAD_LSTR_double), that holds an estimate of the solution **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.

U is an array INTENT(INOUT) argument of dimension m and type default REAL (double precision in GALAHAD_LSTR_double), that is used to hold left-Lanczos vectors used during the iteration. On initial or restart entry, U must contain the vector **b**. If inform%status = 2 or 4 on exit, U must be reset as directed by inform%status; otherwise it must be left unchanged.

V is an array INTENT(INOUT) argument of dimension n and type default REAL (double precision in GALAHAD_LSTR_double), that is used to hold left-Lanczos vectors used during the iteration. It need not be set on initial or restart entry. If inform%status = 3 on exit, V must be reset as directed by inform%status; otherwise it must be left unchanged.

data is a scalar INTENT(INOUT) argument of type LSTR_data_type (see Section 2.1.3). 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 LSTR_initialize.

control is a scalar INTENT(IN) argument of type LSTR_control_type. (see Section 2.1.1). Default values may be assigned by calling LSTR_initialize prior to the first call to LSTR_solve.

inform is a scalar INTENT(INOUT) argument of type LSTR_inform_type (see Section 2.1.2). On initial entry, the component status must be set to 1, while for a restart entry it must be set to 5. The remaining components need not be set. A successful call to LSTR_solve is indicated when the component status has the value 0. For other return values of status, see Sections 2.3 and 2.4.

2.2.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

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

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

control is a scalar INTENT(IN) argument of type LSTR_control_type exactly as for LSTR_solve.

inform is a scalar INTENT(OUT) argument of type LSTR_type exactly as for LSTR_solve. Only the component status will be set on exit, and a successful call to LSTR_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.4.

2.3 Reverse communication

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

2. The user must perform the operation

$$\mathbf{u} := \mathbf{u} + \mathbf{A}\mathbf{v},$$

and recall LSTR_solve. The vectors **u** and **v** are available in the arrays U and V respectively, and the result **u** must overwrite the content of U. No argument except U should be altered before recalling LSTR_solve.

3. The user must perform the operation

$$\mathbf{v} := \mathbf{v} + \mathbf{A}^T\mathbf{u},$$

and recall LSTR_solve. The vectors **u** and **v** are available in the arrays U and V respectively, and the result **v** must overwrite the content of V. No argument except V should be altered before recalling LSTR_solve.

4. The user should reset U to **b** and recall LSTR_solve. No argument except U should be altered before recalling LSTR_solve.

2.4 Warning and error messages

A negative value of inform%status on exit from LSTR_solve or LSTR_terminate indicates that an error has 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. (LSTR_solve only) At least one of the restrictions m > 0, n > 0 or radius > 0 has been violated.

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

-25. (LSTR_solve only) inform%status is not > 0 on entry.

-30. (LSTR_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.

2.5 Weighted least-squares, scaled trust-regions and preconditioning

The package may also be used to solve weighted least-squares problems involving the objective $\|\mathbf{W}(\mathbf{A}\mathbf{x}-\mathbf{b})\|_2$ and a scaled trust region $\|\mathbf{S}\mathbf{x}\|_2 \leq \Delta$ simply by solving instead the problem

$$\min \|\bar{\mathbf{A}}\bar{\mathbf{x}}-\bar{\mathbf{b}}\|_2 \;\text{ subject to }\; \|\bar{\mathbf{x}}\|_2 \leq \Delta,$$

where $\bar{\mathbf{A}} = \mathbf{W}\mathbf{A}\mathbf{S}^{-1}$ and $\bar{\mathbf{b}} = \mathbf{W}\mathbf{b}$ and then recovering $\mathbf{x} = \mathbf{S}^{-1}\bar{\mathbf{x}}$. Note the implication here that $\mathbf{S}$ must be non-singular.

Thus on initial entry (`inform%status` = 1) and re-entry (`inform%status` = 4), `U` should contain $\mathbf{W}\mathbf{b}$, while for `inform%status` = 2 and 3 entries, the operations

$$\mathbf{u} := \mathbf{u} + \mathbf{W}\mathbf{A}\mathbf{S}^{-1}\mathbf{v} \;\text{ and }\; \mathbf{v} := \mathbf{v} + \mathbf{S}^{-T}\mathbf{A}^T\mathbf{W}^T\mathbf{u}$$

respectively, should be performed.

Note that the choice of $\mathbf{W}$ and $\mathbf{S}$ will effect the convergence of the method, and thus good choices may be used to accelerate its convergence. This is often known as preconditioning, but be aware that preconditioning changes the norms that define the problem. Good preconditioners will cluster the singular values of $\bar{\mathbf{A}}$ around a few distinct values, and ideally (but usually unrealistically) all the singular values will be mapped to 1.

2.6 Restart 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 `U` to $\mathbf{b}$, and recalling `LSTR_solve` with `inform%status = 5` and the remaining arguments unchanged. This will determine the best solution within the Krylov space investigated in the previous minimization.

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 `LSTR_control_type` (see Section 2.1.1), by reading an appropriate data specification file using the subroutine `LSTR_read_specfile`. This facility is useful as it allows a user to change `LSTR` control parameters without editing and recompiling programs that call `LSTR`.

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 `LSTR_read_specfile` must start with a "`BEGIN LSTR`" command and end with an "`END`" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by LSTR_read_specfile .. )
  BEGIN LSTR
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by LSTR_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN LSTR" 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 LSTR SPECIFICATION
```

and

```
END LSTR SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN LSTR" 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 LSTR_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 LSTR_read_specfile.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL LSTR_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type LSTR_control_type (see Section 2.1.1). Default values should have already been set, perhaps by calling LSTR_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 2.1.1) of control that each affects are given in Table 2.1.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
minimum-number-of-iterations	%itmin	integer
maximum-number-of-iterations	%itmax	integer
maximum-number-of-boundary-iterations	%itmax_on_boundary	integer
maximum-number-of-inner-iterations	%bitmax	integer
number-extra-n-vectors-used	%extra_vectors	integer
relative-accuracy-required	%stop_relative	real
absolute-accuracy-required	%stop_absolute	real
fraction-optimality-required	%fraction_opt	real
stop-as-soon-as-boundary-encountered	%steihaug_toint	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical

Table 2.1: Specfile commands and associated components of control.

device is a scalar INTENT(IN) argument of type default INTEGER, 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.

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 norm of the residual $\mathbf{A}\mathbf{x}-\mathbf{b}$, the norm of the gradient, and the norm of $\mathbf{x}$. A further message will be printed if the constraint boundary is encountered during the current iteration. Thereafter, the one-line summary will also record the value of the Lagrange multiplier λ and the number of Newton steps required to find λ. 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 "bi-diagonal" least-squares subproblem.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: LSTR_solve calls the BLAS function *ROTG, where * is S for the default real version and D for the double precision version.

Other modules used directly: LSTR_solve calls the GALAHAD packages GALAHAD_SYMBOLS, GALAHAD_SPACE, 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: $m > 0$, $n > 0$, $\Delta > 0$.

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 $\mathbf{A}^T(\mathbf{A}\mathbf{x}-\mathbf{b})+\lambda\mathbf{x}=0$, where $\lambda \geq 0$ is a Lagrange multiplier corresponding to the trust-region constraint $\|\mathbf{x}\|_2 \leq \Delta$.

The method is iterative. Starting with the vector $\mathbf{u}_1 = \mathbf{b}$, a bi-diagonalisation process is used to generate the vectors $\mathbf{v}_k$ and $\mathbf{u}_k + 1$ so that the n by k matrix $\mathbf{V}_k = (\mathbf{v}_1 \ldots \mathbf{v}_k)$ and the m by $(k+1)$ matrix $\mathbf{U}_k = (\mathbf{u}_1 \ldots \mathbf{u}_{k+1})$ together satisfy

$$\mathbf{A}\mathbf{V}_k = \mathbf{U}_{k+1}\mathbf{B}_k \text{ and } \mathbf{b} = \|\mathbf{b}\|\mathbf{U}_{k+1}\mathbf{e}_1$$

where $\mathbf{B}_k$ is $(k+1)$ by k and lower bi-diagonal, $\mathbf{U}_k$ and $\mathbf{V}_k$ have orthonormal columns and $\mathbf{e}_1$ is the first unit vector. The solution sought is of the form $\mathbf{x}_k = \mathbf{V}_k\mathbf{y}_k$, where $\mathbf{y}_k$ solves the bi-diagonal least-squares trust-region problem

$$\min \|\mathbf{B}_k\mathbf{y} - \|\mathbf{b}\|\mathbf{e}_1\|_2 \text{ subject to } \|\mathbf{y}\|_2 \leq \Delta. \tag{4.1}$$

If the trust-region constraint is inactive, the solution $\mathbf{y}_k$ may be found, albeit indirectly, via the LSQR algorithm of Paige and Saunders which solves the bi-diagonal least-squares problem

$$\min \|\mathbf{B}_k\mathbf{y} - \|\mathbf{b}\|\mathbf{e}_1\|_2$$

using a QR factorization of B_k. Only the most recent $\mathbf{v}_k$ and $\mathbf{u}_{k+1}$ are required, and their predecessors discarded, to compute $\mathbf{x}_k$ from $\mathbf{x}_{k-1}$. This method has the important property that the iterates $\mathbf{y}$ (and thus $\mathbf{x}_k$) generated increase in norm with k. Thus as soon as an LSQR iterate lies outside the trust-region, the required solution to (4.1) 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

$$\min \|\mathbf{B}_k\mathbf{y} - \|\mathbf{b}\|\mathbf{e}_1\|_2 \text{ subject to } \|\|\mathbf{y}\|_2 = \Delta, \tag{4.2}$$

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

$$\mathbf{B}_k^T(\mathbf{B}_k\mathbf{y}(\lambda) - \|\mathbf{b}\|\mathbf{e}_1) + \lambda\mathbf{y}(\lambda) = 0 \text{ and } \|\mathbf{y}(\lambda)\|_2 = \Delta$$

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

$$\min \left\| \begin{pmatrix} \mathbf{B}_k \\ \lambda^{\frac{1}{2}}\mathbf{I} \end{pmatrix} y - \|\mathbf{b}\|\mathbf{e}_1 \right\|$$

and may be found efficiently. Given $\mathbf{y}(\lambda)$, Newton's method is then used to find λ_k as the positive root of $\|\mathbf{y}(\lambda)\|_2 = \Delta$. Unfortunately, unlike when the solution lies in the interior of the trust-region, it is not known how to recur $\mathbf{x}_k$ from $\mathbf{x}_{k-1}$ given $\mathbf{y}_k$, and a second pass in which $\mathbf{x}_k = \mathbf{V}_k\mathbf{y}_k$ is regenerated is needed—this need only be done once $\mathbf{x}_k$ 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.

References: 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 are 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).

5 EXAMPLE OF USE

Suppose we wish to solve a problem in 50 unknowns, whose data is

$$\mathbf{A} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ 1 & & & & \\ & 2 & & & \\ & & \ddots & & \\ & & & 50 & \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ 1 \end{pmatrix},$$

with a radius $\Delta = 1$. Suppose in addition that we wish to investigate the solution beyond the Steihaug-Toint point, but are content with an approximation which is within 99% of the best. Then we may use the following code:

```
PROGRAM GALAHAD_LSTR_EXAMPLE  !  GALAHAD 2.4 - 15/05/2010 AT 14:15 GMT
USE GALAHAD_LSTR_DOUBLE                          ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: working = KIND( 1.0D+0 ) ! set precision
REAL ( KIND = working ), PARAMETER :: one = 1.0_working, zero = 0.0_working
INTEGER, PARAMETER :: n = 50, m = 2 * n           ! problem dimensions
INTEGER :: i
REAL ( KIND = working ) :: radius = 1.0_working  ! radius of one
REAL ( KIND = working ), DIMENSION( n ) :: X, V
REAL ( KIND = working ), DIMENSION( m ) :: U, RES
TYPE ( LSTR_data_type ) :: data
TYPE ( LSTR_control_type ) :: control
TYPE ( LSTR_inform_type ) :: inform
CALL LSTR_initialize( data, control, inform ) ! Initialize control parameters
control%steihaug_toint = .FALSE.       ! Try for an accurate solution
control%fraction_opt = 0.99            ! Only require 99% of the best
U = one                                ! The term b is a vector of ones
inform%status = 1
DO                                     ! Iteration to find the minimizer
  CALL LSTR_solve( m, n, radius, X, U, V, data, control, inform )
  SELECT CASE( inform%status )  !  Branch as a result of inform%status
  CASE( 2 )                     !  Form u <- u + A * v
    U( : n ) = U( : n ) + V     !  AˆT = ( I : diag(1:n) )
    DO i = 1, n
      U( n + i ) = U( n + i ) + i * V( i )
    END DO
  CASE( 3 )                     !  Form v <- v + AˆT * u
    V = V + U( : n )
    DO i = 1, n
      V( i ) = V( i ) + i * U( n + i )
    END DO
  CASE ( 4 )                   !  Restart
    U = one                    !  re-initialize u to b
  CASE ( - 30, 0 )             !  Successful return
    RES = one                  !  Compute the residuals for checking
    RES( : n )  = RES( : n ) - X
    DO i = 1, n
      RES( n + i ) = RES( n + i ) - i * X( i )
    END DO
    WRITE( 6, "( 1X, I0, ' 1st pass and ', I0, ' 2nd pass iterations' )" ) &
      inform%iter, inform%iter_pass2
    WRITE( 6, "( '   ||x||  recurred and calculated = ', 2ES16.8 )" )      &
      inform%x_norm, SQRT( DOT_PRODUCT( X, X ) )
    WRITE( 6, "( ' ||Ax-b|| recurred and calculated = ', 2ES16.8 )" )      &
      inform%r_norm, SQRT( DOT_PRODUCT( RES, RES ) )
    CALL LSTR_terminate( data, control, inform ) ! delete internal workspace
    EXIT
  CASE DEFAULT                 !  Error returns
    WRITE( 6, "( ' LSTR_solve exit status = ', I6 ) " ) inform%status
    CALL LSTR_terminate( data, control, inform ) ! delete internal workspace
    EXIT
  END SELECT
END DO
```

```
  END PROGRAM GALAHAD_LSTR_EXAMPLE
```

This produces the following output:

```
 59 1st pass and 28 2nd pass iterations
   ||x||  recurred and calculated =   1.00000000E+00  1.00000000E+00
 ||Ax-b|| recurred and calculated =   6.57514081E+00  6.57514081E+00
```