GALAHAD

L2RT

1 SUMMARY

Given a real m by n matrix $\mathbf{A}$, a real m vector $\mathbf{b}$ and scalars $\sigma > 0$, $\mu \geq 0$ and $p \geq 2$, this package finds an **approximate minimizer of the regularised linear-least-ℓ_2-norm objective function** $\sqrt{\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2} + \frac{1}{p}\sigma\|\mathbf{x}\|_2^p$. This problem commonly occurs as a subproblem in nonlinear optimization calculations involving quadratic or cubic regularisation, and may be used to regularise 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 $\mathbf{u} + \mathbf{Av}$ and $\mathbf{v} + \mathbf{A}^T\mathbf{u}$.

ATTRIBUTES — Versions: `GALAHAD_L2RT_single`, `GALAHAD_L2RT_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_NORMS`, `GALAHAD_ROOTS`, `GALAHAD_LSTR`, `GALAHAD_SPECFILE`, `*ROTG`. **Date:** January 2008. **Origin:** N. I. M. Gould, Oxford University and Rutherford Appleton Laboratory and M. Porcelli, Università degli Studi di Firenze. **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_L2RT_single`

Double precision version

> `USE GALAHAD_L2RT_double`

If it is required to use both modules at the same time, the derived types `L2RT_control_type`, `L2RT_inform_type`, `L2RT_data_type`, (Section 2.1) and the subroutines `L2RT_initialize`, `L2RT_solve`, `L2RT_terminate` (Section 2.2) and `L2RT_read_specfile` (Section 2.6) 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 `L2RT_control_type` is used to hold controlling data. Default values may be obtained by calling `L2RT_initialize` (see Section 2.2.1). The components of `L2RT_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 `L2RT_solve` and `L2RT_terminate` is suppressed if `error` ≤ 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 `L2RT_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` ≤ 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`.

itmin is a scalar variable of type default INTEGER, that holds the minimum number of iterations which will be performed by L2RT_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 L2RT_solve. If itmax is set to a negative number, it will be reset by L2RT_solve to $\max(m,n)+10$. The default is itmax = -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 L2RT_solve. If bitmax is set to a negative number, it will be reset by L2RT_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 during the second pass. The default is extra_vectors = 0.

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_L2RT_double), that holds the relative and absolute convergence tolerances (see Section 4). The computed solution $\mathbf{x}$ is accepted by L2RT_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 the multiplier $\lambda = \mu + \sigma\|\mathbf{x}\|_2^{p-2}\sqrt{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2}$. The defaults are stop_relative = $\sqrt{u}$ and stop_absolute = 0.0, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_L2RT_double).

fraction_opt is a scalar variable of type default REAL (double precision in GALAHAD_L2RT_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 L2RT_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of L2RT_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.

---

obj is a scalar variable of type default REAL (double precision in GALAHAD_L2RT_double), that holds the value of the objective function $\sqrt{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2} + \frac{1}{p}\sigma\|\mathbf{x}\|_2^p$.

multiplier is a scalar variable of type default REAL (double precision in GALAHAD_L2RT_double), that holds the value of the multiplier $\lambda = \mu + \sigma\|\mathbf{x}\|_2^{p-2}\sqrt{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2}$.

x_norm is a scalar variable of type default REAL (double precision in GALAHAD_L2RT_double), that holds the current value of $\|\mathbf{x}\|_2$.

r_norm is a scalar variable of type default REAL (double precision in GALAHAD_L2RT_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_L2RT_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 L2RT_data_type is used to hold all the data for a particular problem between calls of L2RT procedures. This data should be preserved, untouched, from the initial call to L2RT_initialize to the final call to L2RT_terminate.

2.2 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.6 for further features):

1. The subroutine L2RT_initialize is used to set default values, and initialize private data.

2. The subroutine L2RT_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 L2RT_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by L2RT_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 L2RT_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 L2RT_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type L2RT_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 L2RT_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 L2RT_initialize.

inform is a scalar INTENT(OUT) argument of type L2RT_inform_type (see Section 2.1.2). A successful call to L2RT_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 L2RT_solve(m, n, p, sigma, mu, 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:** $n > 0$.

p   is a scalar INTENT(IN) variable of type default REAL (double precision in GALAHAD_L2RT_double), that must be set on initial entry to the desired order p of regularisation. **Restriction:** $p \geq 2$.

sigma is a scalar INTENT(IN) variable of type default REAL (double precision in GALAHAD_L2RT_double), that must be set on initial entry to the value of the weight σ associated with the regularisation term. **Restriction:** $\sigma > 0$.

mu   is a scalar INTENT(IN) variable of type default REAL (double precision in GALAHAD_L2RT_double), that must be set on initial entry to the value of the shift μ. **Restriction:** $\mu \geq 0$.

X   is an array INTENT(INOUT) argument of dimension n and type default REAL (double precision in GALAHAD_L2-RT_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_L2-RT_double), that is used to hold left-Lanczos vectors used during the iteration. On initial 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_L2-RT_double), that is used to hold left-Lanczos vectors used during the iteration. It need not be set on initial 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 L2RT_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 L2RT_initialize.

control is a scalar INTENT(IN) argument of type L2RT_control_type. (see Section 2.1.1). Default values may be assigned by calling L2RT_initialize prior to the first call to L2RT_solve.

inform is a scalar INTENT(INOUT) argument of type L2RT_inform_type (see Section 2.1.2). On initial entry, the component status must be set to 1. The remaining components need not be set. A successful call to L2-RT_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 L2RT_terminate(data, control)

data is a scalar INTENT(INOUT) argument of type L2RT_data_type exactly as for L2RT_solve that must not have been altered **by the user** since the last call to L2RT_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type L2RT_control_type exactly as for L2RT_solve.

All use is subject to the conditions of the GNU Lesser General Public License version 3.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

inform is a scalar `INTENT(OUT)` argument of type `L2RT_type` exactly as for `L2RT_solve`. Only the component `status` will be set on exit, and a successful call to `L2RT_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 `L2RT_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 `L2RT_solve`. The vectors $\mathbf{u}$ and $\mathbf{v}$ are available in the arrays U and V respectively, and the result $\mathbf{u}$ must overwrite the content of U. No argument except U should be altered before recalling `L2RT_solve`.

3. The user must perform the operation
$$\mathbf{v} := \mathbf{v} + \mathbf{A}^T\mathbf{u},$$
and recall `L2RT_solve`. The vectors $\mathbf{u}$ and $\mathbf{v}$ are available in the arrays U and V respectively, and the result $\mathbf{v}$ must overwrite the content of V. No argument except V should be altered before recalling `L2RT_solve`.

4. The user should reset U to $\mathbf{b}$ and recall `L2RT_solve`. No argument except U should be altered before recalling `L2RT_solve`.

2.4 Warning and error messages

A negative value of `inform%status` on exit from `L2RT_solve` or `L2RT_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. (`L2RT_solve` only) At least one of the restrictions m > 0, n > 0, sigma > 0, mu $\geq$ 0 or p $\geq$ 2 has been violated.

-18. (`L2RT_solve` only) More than `control%itmax` iterations have been performed without obtaining convergence.

-25. (`L2RT_solve` only) `inform%status` is not > 0 on entry.

2.5 Weighted least-ℓ_2-norms, scaled regularisation and preconditioning

The package may also be used to solve a weighted regularisation least-ℓ_2-norm problem
$$\min \|\mathbf{W}(\mathbf{A}\mathbf{x} - \mathbf{b})\|_2 + \tfrac{1}{p}\sigma\|\mathbf{S}\mathbf{x}\|_2^p$$
simply by solving instead the problem
$$\min \|\bar{\mathbf{A}}\bar{\mathbf{x}} - \bar{\mathbf{b}}\|_2 + \tfrac{1}{p}\sigma\|\bar{\mathbf{x}}\|_2^p$$
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 **Wb**, while for `inform%status = 2` and 3 entries, the operations

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

respectively, should be performed.

Note that the choice of **W** and **S** will affect 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 **Ā** around a few distinct values, and ideally (but usually unrealistically) all the singular values will be mapped to 1.

2.6 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable `control` of type `L2RT_control_type` (see Section 2.1.1), by reading an appropriate data specification file using the subroutine `L2RT_read_specfile`. This facility is useful as it allows a user to change `L2RT` control parameters without editing and recompiling programs that call `L2RT`.

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 `L2RT_read_specfile` must start with a "BEGIN L2RT" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by L2RT_read_specfile .. )
  BEGIN L2RT
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by L2RT_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN L2RT" 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 L2RT SPECIFICATION
```

and

```
  END L2RT SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN L2RT" 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 L2RT_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 L2RT_read_specfile.

2.6.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL L2RT_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type L2RT_control_type (see Section 2.1.1). Default values should have already been set, perhaps by calling L2RT_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-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
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.

2.7 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 $\mathbf{Ax} - \mathbf{b}$, 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 $\lambda = \mu + \sigma\|\mathbf{x}\|_2^{p-2}\sqrt{\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2}$, 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: L2RT_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: L2RT_solve calls the GALAHAD packages GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD_NORMS, GALAHAD_ROOTS, GALAHAD_LSTR and GALAHAD_SPECFILE.

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

Restrictions: $m > 0$, $n > 0$, $\sigma > 0$, $\mu \geq 0$, $p \geq 2$.

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

$$\lambda = \mu + \sigma\|\mathbf{x}\|_2^{p-2}\sqrt{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2}.$$

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

$$\mathbf{A}\mathbf{V}_k = \mathbf{U}_{k+1}\mathbf{B}_k \text{ and } \mathbf{b} = \|\mathbf{b}\|_2\mathbf{U}_{k+1}\mathbf{e}_1$$

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 regularised least-squares problem

$$\min \|\mathbf{B}_k\mathbf{y} - \|\mathbf{b}\|\mathbf{e}_1\|_2 + \tfrac{1}{p}\sigma\|\mathbf{y}\|_2^p. \tag{4.1}$$

To minimize (4.1), the optimality conditions

$$(\mathbf{B}_k^T(\mathbf{B}_k\mathbf{y}(\lambda) - \|\mathbf{b}\|\mathbf{e}_1) + \lambda\mathbf{y}(\lambda) = 0, \tag{4.2}$$

where

$$\lambda = \mu + \sigma\|\mathbf{y}(\lambda)\|_2^{p-2}\sqrt{\|\mathbf{B}_k\mathbf{y}(\lambda) - \|\mathbf{b}\|\mathbf{e}_1\|_2^2 + \mu\|\mathbf{y}(\lambda)\|_2^2},$$

are used as the basis of an iteration. The vector $\mathbf{y}(\lambda)$ is equivalently the solution to the regularised least-squares problem

$$\min \left\|\begin{pmatrix} \mathbf{B}_k \\ \lambda^{\frac{1}{2}}\mathbf{I} \end{pmatrix} y - \|\mathbf{b}\|\mathbf{e}_1\right\|_2. \tag{4.3}$$

Thus, given an estimate $\lambda \geq 0$, (4.3) may be efficiently solved to give $\mathbf{y}(\lambda)$. It is then simply a matter of adjusting λ (for example by a Newton-like process) to solve the scalar nonlinear equation

$$\theta(\lambda) \equiv \sigma\|\mathbf{y}(\lambda)\|_2^{p-2}\sqrt{\|\mathbf{B}_k\mathbf{y}(\lambda) - \|\mathbf{b}\|\mathbf{e}_1\|_2^2 + \mu\|\mathbf{y}(\lambda)\|_2^2} + \mu - \lambda = 0. \tag{4.4}$$

In practice (4.4) is reformulated, and a more rapidly converging iteration is used. Having found $\mathbf{y}_k$, 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

---

All use is subject to the conditions of the GNU Lesser General Public License version 3.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

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 the equation (4.4) significantly simplifies.

References: A complete description of the underlying quadratically-regularised least-squares problem 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 the Newton-like process needed to determine λ and other details are described in

C. Cartis, N. I. M. Gould and Ph. L. Toint, Trust-region and other regularisations 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

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

with regularisation weight $\sigma = 1$, shift $\mu = 0$ and order $p = 3$. Suppose further that we are content with an approximation which is within 99% of the best. Then we may use the following code:

```
PROGRAM GALAHAD_L2RT_EXAMPLE   !  GALAHAD 2.4 - 15/05/20010 AT 14:15 GMT
USE GALAHAD_L2RT_DOUBLE                      ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: working = KIND( 1.0D+0 )     ! set precision
REAL ( KIND = working ), PARAMETER :: one = 1.0_working
INTEGER, PARAMETER :: n = 50, m = 2 * n           ! problem dimensions
INTEGER :: i
REAL ( KIND = working ) :: p = 3.0_working        ! order of regulatisation
REAL ( KIND = working ) :: sigma = 1.0_working    ! regulatisation weight
REAL ( KIND = working ) :: mu = 0.0_working       ! zero shift
REAL ( KIND = working ), DIMENSION( n ) :: X, V
REAL ( KIND = working ), DIMENSION( m ) :: U, RES
TYPE ( L2RT_data_type ) :: data
TYPE ( L2RT_control_type ) :: control
TYPE ( L2RT_inform_type ) :: inform
CALL L2RT_initialize( data, control, inform ) ! Initialize control parameters
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 L2RT_solve( m, n, p, sigma, mu, 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      !  A^T = ( I : diag(1:n) )
        DO i = 1, n
          U( n + i ) = U( n + i ) + i * V( i )
        END DO
      CASE( 3 )                       !  Form v <- v + A^T * 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 ( 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, "( ' objective recurred and calculated = ', 2ES16.8 )" )    &
          inform%obj, SQRT( DOT_PRODUCT( RES, RES ) +                         &
           mu * SQRT( DOT_PRODUCT( X, X ) ) ** 2 ) + ( sigma / p ) *          &
          ( SQRT( DOT_PRODUCT( X, X ) ) ) ** p
        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 L2RT_terminate( data, control, inform ) ! delete internal workspace
        EXIT
      CASE DEFAULT                    !  Error returns
        WRITE( 6, "( ' L2RT_solve exit status = ', I6 ) " ) inform%status
        CALL L2RT_terminate( data, control, inform ) ! delete internal workspace
        EXIT
      END SELECT
   END DO
   END PROGRAM GALAHAD_L2RT_EXAMPLE
```

This produces the following output:

```
 58 1st pass and 19 2nd pass iterations
 objective recurred and calculated =   6.79093482E+00  6.79093482E+00
    ||x||  recurred and calculated =   6.55966193E-01  6.55966193E-01
  ||Ax-b|| recurred and calculated =   6.69684923E+00  6.69684923E+00
```