1 SUMMARY

FILTRANE is a package for solving the general smooth feasibility problem, that is the problem to find a “feasible” vector \( x \in \mathbb{R}^n \) such that

\[
\mathbf{c}' \leq \mathbf{c}(x) \leq \mathbf{c}'',
\]

and

\[
\mathbf{x}' \leq x \leq \mathbf{x}'',
\]

where \( \mathbf{c}(x) \) is a smooth function from \( \mathbb{R}^n \) into \( \mathbb{R}^m \) and where inequalities are understood componentwise. The vectors \( \mathbf{c}' \leq \mathbf{c}'' \) and \( \mathbf{x}' \leq \mathbf{x}'' \) are \( m \)- and \( n \)-dimensional, respectively, and may contain components equal to minus or plus infinity. For a given \( i \) between 1 and \( m \), it is assumed that the \( i \)-th component of either \( \mathbf{c}' \) or \( \mathbf{c}'' \) is finite, while, for \( j \) between 1 and \( n \), the \( j \)-th components of both \( \mathbf{x}' \) and \( \mathbf{x}'' \) are allowed to be infinite. In what follows, we will say that variable \( j \) is bounded if the \( j \)-th component of either \( \mathbf{x}' \) or \( \mathbf{x}'' \) is finite. Equalities may be specified by choosing identical lower and upper bounds on either \( \mathbf{c} \) or \( x \). The above framework therefore covers all combinations of bounds with linear/nonlinear equalities and/or inequalities.

If a feasible point cannot be found, it is desired to find a local minimizer of the Euclidean norm \( \| \cdot \| \) of the constraints violations, that is to find a local minimizer of the function

\[
\min_{x} \frac{1}{2} \| \theta(x) \|^2,
\]

where we define

\[
\theta(x) \equiv \left( \frac{\max(|\mathbf{c}' - \mathbf{c}(x)|_+, |\mathbf{c}(x) - \mathbf{c}''|_+)}{\max(|\mathbf{x}' - \mathbf{x}|_+, |\mathbf{x} - \mathbf{x}''|_+)} \right) \in \mathbb{R}^p,
\]

where, for a vector \( y \), \( |y|_+ = \max(0, y) \), and where all maxima are taken componentwise.

FILTRANE also allows the partitioning the problem's constraints and bounded variables into \( p \) (not necessarily disjoint) sets or “groups”. In this case, each \( \theta(x) \) has \( p \) components defined as

\[
\theta_{[\ell]}(x) \equiv \left( \frac{\max(|\mathbf{c}_{[\ell]} - \mathbf{c}_{[\ell]}(x)|_+, |\mathbf{c}_{[\ell]}(x) - \mathbf{c}'_{[\ell]}|_+)}{\max(|\mathbf{x}_{[\ell]} - \mathbf{x}_{[\ell]}|_+, |\mathbf{x}_{[\ell]} - \mathbf{x}''_{[\ell]}|_+)} \right) \text{ for } \ell = 1, \ldots, p,
\]

where the subscript \([\ell]\) indicates that only components belonging to the \( \ell \)-th group are considered.

ATTRIBUTES — Versions: GALAHAD_FILTRANE_single, GALAHAD_FILTRANE_double.


2 HOW TO USE THE PACKAGE

Access to the package requires a USE statement such as

Single precision version

USE GALAHAD_FILTRANE_single

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.
Double precision version
USE GALAHAD_FILTRANE_double

If it is required to use both modules at the same time, the derived types NLPT_problem_type, FILTRANE_control_type, FILTRANE_inform_type and FILTRANE_data_type (Section 2.3) and the four subroutines FILTRANE_initialize, FILTRANE_read_specfile, FILTRANE_solve, FILTRANE_terminate, (Section 2.4) must be renamed on one of the USE statements.

2.1 Matrix storage formats
The constraint Jacobian matrix $J(x)$ must be stored in sparse co-ordinate format. In this format, only the nonzero entries of the matrices are stored. For the $l$-th entry of $J(x)$, its row index $i$, column index $j$ and value $J_{ij}$ are stored in the $l$-th components of the integer arrays J_row, J_col and real array J_val. The order is unimportant, but the total number of entries J_size is also required.

2.2 The GALAHAD symbols
The following description make use of “symbols” that are publicly available in the GALAHAD_SYMBOLS module. These symbols are conventional names given to specific integer values, that allow a more natural specification of the various options and parameters of the package. Each symbol provided in the SYMBOLS module is of the form GALAHAD_NAME, where NAME is the name of the symbol. For clarity and conciseness, we will represent such a symbol by GALAHAD_NAME (in sans-serif upper case font) in what follows. See Section 5 to see how symbols may be used in the program unit that calls the FILTRANE subroutines.

2.3 The derived data types
In addition to the problem data type, three derived data types are accessible from the package.

2.3.1 The problem type
The derived data type NLPT_problem_type is used to hold the problem: we refer the reader to the documentation of the GALAHAD_NLPT module for a full description. We only consider here the components that are of interest in conjunction with FILTRANE.

We first review the components of the problem data type which must be set on entry in FILTRANE_solve.

- $n$ is a scalar variable of type default INTEGER, that holds the number of problem variables, $n$.
- $m$ is a scalar variable of type default INTEGER, that holds the number of problem constraints, $m$.
- $x$ is a rank-one allocatable array of dimension $n$ and type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the values $x$ of the problem variables at the initial point. The $j$-th component of $x$, $j = 1, \ldots, n$, contains $x_j$.
- $x_l$ is a rank-one allocatable array of dimension $n$ and type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the vector of lower bounds $x^l$ on the problem variables. The $j$-th component of $x_l$, $j = 1, \ldots, n$, contains $x^l_j$. Infinite bounds are allowed by setting the corresponding components of $x_l$ to any value smaller than that $-\infty$.
- $x_u$ is a rank-one allocatable array of dimension $n$ and type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the vector of upper bounds $x^u$ on the problem variables. The $j$-th component of $x_u$, $j = 1, \ldots, n$, contains $x^u_j$. Infinite bounds are allowed by setting the corresponding components of $x_u$ to any value larger than that $\infty$. 

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.
c_l is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the vector of lower bounds c_l on the general constraints. The i-th component of c_l, \( i = 1, \ldots, m \), contains c_{l i}. Infinite bounds are allowed by setting the corresponding components of c_l to any value smaller than -\infinity.

c_u is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the vector of upper bounds c_u on the general constraints. The i-th component of c_u, \( i = 1, \ldots, m \), contains c_{u i}. Infinite bounds are allowed by setting the corresponding components of c_u to any value larger than infinity.

J_size is a scalar variable of type default INTEGER, which holds the number of nonzero entries of the constraints Jacobian matrix.

J_type is a scalar variable of type default INTEGER, which specifies the format in which the constraints Jacobian is stored. Only the value GALAHAD_COORDINATE is allowed as this is the only storage scheme provided within FILTRANE.

infinity is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), which holds the value such that real numbers with absolute value less than infinity are finite, all others being considered infinite.

In addition, the following components must be allocated:

x_status is a rank-one allocatable array of dimension n and type default INTEGER,

c is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_FILTRANE_double),

y is a rank-one allocatable array of dimension m and type default REAL (double precision in GALAHAD_FILTRANE_double),

g is a rank-one allocatable array of dimension n and type default REAL (double precision in GALAHAD_FILTRANE_double),

equation is a rank-one allocatable array of dimension n and type default LOGICAL,

J_val is a rank-one allocatable array of dimension J_size + n and type default REAL (double precision in GALAHAD_FILTRANE_double),

J_row is a rank-one allocatable array of dimension J_size + n and type default INTEGER,

J_col is a rank-one allocatable array of dimension J_size + n and type default INTEGER.

However, they need not be assigned a value.

Additionally, the arrays x_l, x_u and x_status need not be allocated if there is no bound on the problem’s variables.

Similarly, the arrays c, c_l, c_u, y, J_val, J_row and J_col need not be allocated if there are no constraints (i.e. m = 0), in which case the values of J_ne and J_type are also irrelevant. Furthermore, J_val, J_row and J_col need not be allocated if m > 0 and external Jacobian products are requested (see Section 2.3.2), in which case the values of J_ne and J_type are again irrelevant.

2.3.2 The derived data type for holding control parameters

The derived data type FILTRANE_control_type is used to hold controlling data. Default values may be obtained by calling FILTRANEInitialize (see Section 2.4.1), while individual components may also be changed by calling FILTRANE_read_specfile (see Section 2.7.1). The components of FILTRANE_control_type are:

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.
c_accuracy is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that specifies an accuracy threshold such that the FILTRANE iteration is successfully terminated if each constraint violation is under the threshold. The default is c_accuracy = 10^{-4} in single precision, and c_accuracy = 10^{-6} in double precision.

g_accuracy is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that specifies an accuracy threshold such that the FILTRANE iteration is successfully terminated if the (possibly preconditioned) Euclidean norm of \( \nabla_c \theta \) is under the threshold. The default is g_accuracy = 10^{-4} in single precision, and g_accuracy = 10^{-6} in double precision.

stop_on_prec_g is a scalar variable of type default LOGICAL, that has the value .TRUE. iff the preconditioned gradient must be used in the gradient termination test, and has the value .FALSE. iff the unpreconditioned gradient must be used instead. The default is stop_on_prec_g = .TRUE..

stop_on_g_max is a scalar variable of type default LOGICAL, that has the value .TRUE. iff the maximum norm of the gradient must be used in the gradient termination test, and has the value .FALSE. iff the Euclidean norm must be used instead. Requiring the use of the maximum norm is only possible for prec_used = GALAHAD_NONE or for prec_used = GALAHAD_BANDED with semi_bandwidth = 0. It is reset to .FALSE. in all other cases. The default is stop_on_g_max = .FALSE..

max_iterations is a scalar variable of type default INTEGER, that determines the maximum number of iterations of the filter-trust-region algorithm during a call to FILTRANE_solve. If negative, no upper limit is imposed on the number of iterations. The default is max_iterations = 1000.

max_cg_iterations is a scalar variable of type default INTEGER, such that max_cg_iterations times the number of problem’s variables is the maximum number of Lanczos-conjugate-gradients iterations of the Generalized Lanczos Trust-Region subproblems solver at each iteration of filter-trust-region algorithm during a call to FILTRANE_solve. The default is max_cg_iterations = 15.

grouping is a scalar variable of type default INTEGER, that holds the type of equations/inequalities/bounds groups required for the filter tests in the FILTRANE algorithm. Valid values are:

GALAHAD_NONE: each equation is considered individually in the filter,
GALAHAD_AUTOMATIC: the automatic constraints grouping strategy provided by the package is to be used,
GALAHAD_USER_DEFINED: the constraints groups are defined by the user and specified in group.

The default is grouping = GALAHAD_NONE.

nbr_groups is a scalar variable of type default INTEGER, that specifies the number of constraints groups. If grouping is set to GALAHAD_AUTOMATIC, and nbr_groups is positive, the smallest between this value and the problem’s number of variables is used as the number of groups. If grouping is set to GALAHAD_AUTOMATIC, and nbr_groups is negative, the number of constraints groups is set to the number of constraints plus the number of bounded variables divided by the absolute value of the nbr_groups. If grouping is set to GALAHAD_USER_DEFINED, nbr_groups is the number of user-defined constraints groups specified in group. The parameter nbr_groups is not referenced if grouping is GALAHAD_NONE.

group is a pointer to a vector of type default INTEGER of dimension equal to m plus the number of bounded variables. It is only referenced if grouping is set to GALAHAD_USER_DEFINED. In this case, it must be allocated and group(i), its i-th component, for i between 1 and m, specifies the group to which the i-th constraint belongs. Its (m + \ell)-th component specifies the group to which the \ell-th bounded variable belong (bounded variables are numbered consecutively by increasing index, skipping unbounded variables). All its components must be lie between 1 and nbr_groups. It is nullified by FILTRANE if grouping is not set to GALAHAD_USER_DEFINED.
balance_group_values is a scalar variable of type default LOGICAL, that has the value .TRUE. iff the constraints values (at the initial point) must be sorted before they are distributed into groups. This has the effect of approximately balancing the constraint violations (at the initial point) between the groups. It is relevant only if grouping is set to GALAHAD_AUTOMATIC. The default is balance_group_values = .FALSE..

prec_used is a scalar variable of type default INTEGER, that indicates which preconditioning strategy must be used. Valid values are:

GALAHAD_NONE: no preconditioning,
GALAHAD_BANDED: a band preconditioner must be used, where the band has semi-bandwidth semi_bandwidth and is extracted from the Hessian of \( \theta(x) \) (and possibly modified to ensure positive-definiteness),
USER_DEFINED: a user-defined preconditioner for the model’s Hessian must be used, which is applied outside FILTRANE via the reverse communication interface (see Section 2.5).

The default is prec_used = GALAHAD_NONE.

semi_bandwidth is a scalar variable of type default INTEGER, that specifies the semi-bandwidth of the banded preconditioner, in the case where prec_used is set to GALAHAD_BANDED. It must lie between 0 (diagonal preconditioning) and \( n \) (full preconditioning). The default is semi_bandwidth = 5.

external_J_products is a scalar variable of type default LOGICAL, that specifies whether (external_J_products = .FALSE.) the Jacobian is passed as a matrix to FILTRANE, which then computes the products of this matrix or its transpose times any vector internally, or if these products are to be computed outside the package via the reverse communication interface (external_J_products = .TRUE., see Section 2.5), in which case the Jacobian is not used at all inside FILTRANE.

out is a scalar variable of type default INTEGER, that holds the unit number associated with the device used for normal output. The default is out = 6.

errout is a scalar variable of type default INTEGER, that holds the unit number associated with the device used for error output. The default is errout = 6.

print_level is a scalar variable of type default INTEGER, that holds the level of printout requested by the user. See Section 2.8. The default is print_level = GALAHAD_SILENT.

start_print is a scalar variable of type default INTEGER, that holds the index of the first FILTRANE iteration at which printing must occur. The default is start_print = 0 (print from initialization on).

stop_print is a scalar variable of type default INTEGER, that holds the index of the last FILTRANE iteration at which printing must occur. If negative, printing does not stop once started. The default is stop_print = -1 (always print once started).

model_type is a scalar variable of type default INTEGER, that holds the type of model to be used by FILTRANE for the objective function. Valid values are:

GALAHAD_GAUSS_NEWTON: the Gauss-Newton model must be used,
GALAHAD_NEWTON: the full Newton model (including constraints curvatures) must be used,
GALAHAD_AUTOMATIC: an adaptive choice is to be made by FILTRANE between the Gauss-newton and Newton models, based on their respective past performance in terms or prediction or decrease.

The default is model_type = GALAHAD_AUTOMATIC.

model_inertia is a scalar variable of type default INTEGER, that holds the number of past iterations to consider for determining the actual model used. It must be at least 1 and is only relevant if model_type is set to GALAHAD_AUTOMATIC. The default is model_inertia = 5.
model\_criterion is a scalar variable of type default INTEGER, that specifies the criterion to apply for the automatic model selection. Valid values are:

- GALAHAD\_BEST\_FIT: the model is preferred whose prediction of the objective function value is most accurate,
- GALAHAD\_BEST\_REDUCTION: the model is preferred whose use leads to a larger objective function reduction.

The default is model\_criterion = GALAHAD\_BEST\_FIT.

inequality\_penalty\_type is a scalar variable of type default INTEGER, that specifies the type of penalty function used to measure constraint violations. Valid values are:

- 2: the \( \ell_2 \) penalty function is used,
- 3: the \( \ell_3 \) penalty function is used,
- 4: the \( \ell_4 \) penalty function is used.

The default is inequality\_penalty\_type = 2, which corresponds to the use of the Euclidean norm in (1.1).

subproblem\_accuracy is a scalar variable of type default INTEGER, that specifies the type of accuracy requirement for stopping the trust-region subproblem solution. Valid values are

- GALAHAD\_ADAPTIVE: the (possibly preconditioned) norm of the residual for the model must be at most

\[
\min \left[ \varepsilon_1, \|r_0\|^{\varepsilon_2} \cdot \|r_0\| \right]
\]

where \( \|r_0\| \) is the (possibly preconditioned) norm of the residual at the current iterate;
- GALAHAD\_FULL: the (possibly preconditioned) norm of the residual for the model must be at most square root of the machine precision times the (possibly preconditioned) norm of the residual at the current iterate.

The default is subproblem\_accuracy = GALAHAD\_ADAPTIVE.

min\_gltr\_accuracy is a scalar variable of type default REAL (double precision in GALAHAD\_FILTRANE\_double), that holds the minimum relative accuracy \( \varepsilon_1 \) in the accuracy requirement (2.1) for the subproblem solution. It is only relevant if subproblem\_accuracy is set to GALAHAD\_ADAPTIVE, in which case it must be strictly between zero and one. The default is min\_gltr\_accuracy = 0.01.

gltr\_accuracy\_power is a scalar variable of type default REAL (double precision in GALAHAD\_FILTRANE\_double), that holds the power \( \varepsilon_2 \) at which the current residual norm is raised in the accuracy requirement (2.1) for the subproblem solution. It is only relevant if subproblem\_accuracy is set to GALAHAD\_ADAPTIVE, in which case it must be positive. The default is gltr\_accuracy\_power = 1.0.

use\_filter is a scalar variable of type default INTEGER, that specifies when the filter criterion must be used to accept new trial points. Valid values are:

- GALAHAD\_NEVER: the filter must not be used (resulting in a pure trust-region method),
- GALAHAD\_INITIAL: the filter is used as long as trial points are accepted, but is no longer used after a first trial point has been rejected,
- GALAHAD\_ALWAYS: the filter must be used at every iteration.

The default is use\_filter = GALAHAD\_ALWAYS.

filter\_sign\_restriction is a scalar variable of type default LOGICAL, whose value is .TRUE. iff the filter must be constructed by considering the absolute value of the constraints/bounds violations. The default is filter\_sign\_restriction = .FALSE..
maximal_filter_size is a scalar variable of type default LOGICAL, that holds the maximum number of points that the filter can hold. Once this maximum is attained, no further point can be acceptable for the filter and the algorithm reduces to a pure trust-region scheme. If set to a negative value, no upper limit is set on the number of filter entries. The default is maximal_filter_size = -1.

filter_size_increment is a scalar variable of type default INTEGER, that holds the initial filter size (if used), and is also used as an increment for the case where the filter capacity (in memory) must be extended. The default is filter_size_increment = 50.

remove-dominated is a scalar variable of type default LOGICAL, whose value is .TRUE. iff FILTRANE is to remove the dominated filter entries. Setting this parameter to .FALSE. marginally speeds up inclusion of new filter points, at the expense of increased memory requirements and a (marginally) slower acceptance test. The default is remove-dominated = .TRUE..

margin_type is a scalar variable of type default INTEGER, that specifies the quantity that is used to determine the width of the filter margin. Valid values are:

- GALAHAD_CURRENT: the norm of the violations at the current iterate is used,
- GALAHAD_FIXED: the norm of the violations at the filter point itself is used,
- GALAHAD_SMALLEST: the smallest of these two norms is used.

The default is margin_type = GALAHAD_CURRENT.

gamma_f is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the value of the constant defining the filter margin. The default is gamma_f = 0.001.

itr_relax is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the value of the initial trust-region relaxation factor, that is the factor by which the trust-region constraint is relaxed during the initial sequence of unrestricted step. The default is itr_relax = 10^{30}.

str_relax is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the value of the secondary trust-region relaxation factor, that is the factor by which the trust-region constraint is relaxed after a first restricted step has been encountered. The default is str_relax = 1000.

weak_accept_power is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the power $\alpha_2$ in the "weak acceptance criterion" that accepts a trial point $x^+_k$ if

$$\theta(x_k) - \theta(x^+_k) \geq \alpha_1 \min\{1, \theta(x_k)^{\alpha_2}\}. \quad (2.2)$$

This test weakens the filter/trust-region criteria by also accepting steps that produces sufficient descent. If weak_accept_power < 0, this test is not used. The default is weak_accept_power = 2.0.

min_weak_accept_factor is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the parameter $\alpha_1$ in (2.2). It is only relevant if weak_accept_power $\geq$ 0, in which case it must be strictly positive. The default is min_weak_accept_factor = 0.1.

initial_radius is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the initial trust-region radius. It is only relevant if use_filter is different from GALAHAD_NEVER. The default is initial_radius = 1.0.

eta_1 is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the minimum ratio of achieved to predicted reduction for declaring a FILTRANE iteration successful. The default is eta_1 = 0.01.
eta_2 is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the minimum ratio of achieved to predicted reduction for declaring a FILTRANE iteration very successful. The default is $\eta_2 = 0.9$.

gamma_0 is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the strongest factor by which the trust-region radius is decreased when the ratio of achieved to predicted reduction is negative. The default is $\gamma_0 = 0.0625$.

gamma_1 is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the factor by which the trust-region radius is decreased when the FILTRANE iteration is unsuccessful. The default is $\gamma_1 = 0.25$.

gamma_2 is a scalar variable of type default REAL (double precision in GALAHAD_FILTRANE_double), that holds the factor by which the trust-region radius is increased when the FILTRANE iteration is very successful. The default is $\gamma_2 = 2.0$.

save_best_point is a scalar variable of type default LOGICAL, whose value is .TRUE. iff the best point found so far must be saved to be returned as the final iterate. This is only relevant when use_filter is different from GALAHAD_NEVER and requires the storage of an additional vector of dimension $n$. The default is save_best_point = .FALSE..

checkpoint_freq is a scalar variable of type default INTEGER, that holds the frequency (expressed in number of iterations) at which the current values of the problem’s variables and the trust-region radius are saved on a checkpointing file for a possible package restart. It must be non-negative. The default is checkpoint_freq = 0 (no checkpointing).

checkpoint_file is a scalar variable of type default CHARACTER of length 30, that holds the name of the file used for storing checkpointing information on disk. The default is checkpoint_file = FILTRANE.sav.

checkpoint_dev is a scalar variable of type default INTEGER, that holds the number of the device that must be used for input/output of checkpointing operations. The default is checkpoint_dev = 55.

restart_from_checkpoint is a scalar variable of type default LOGICAL, whose value is .TRUE. iff the initial point and constraints values must be read from the checkpointing file checkpoint_file, overriding the input value of problem$x. The default is restart_from_checkpoint = .FALSE..

2.3.3 The derived data type for holding informational parameters

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

status is a scalar variable of type default INTEGER, that gives the exit status of the algorithm. See Sections 2.6 and 2.8 for details.

message is a character array of 3 lines of 80 characters each, containing a description of the exit condition on exit, typically including more information than contained in status. It is printed out on device errout at the end of execution unless print_level is GALAHAD_SILENT.

nbr_iterations is a scalar variable of type default INTEGER, that gives the the final number of FILTRANE iterations.

nbr_cg_iterations is a scalar variable of type default INTEGER, that gives the the final number of iterations performed by GALAHAD_GLTR in solving the subproblems on all successive FILTRANE iterations.

nbr_c_evaluations is a scalar variable of type default INTEGER, that gives the final number of evaluations of the constraints values.

nbr_J_evaluations is a scalar variable of type default INTEGER, that gives the final number of Jacobian evaluations.
2.3.4 The derived data type for holding problem data

The derived data type FILTRANE_data_type is used to hold all the data for a the current problem between calls of FILTRANE procedures. This data should be preserved, untouched, from the initial call to FILTRANE_initialize to the final call to FILTRANE_terminate, except for components that have to be set in the reverse communication interface (see Section 2.5).

2.4 Argument lists and calling sequences

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

1. The subroutine FILTRANE_initialize is used to set default values, and initialize private data.
2. The routine FILTRANE_read_specfile is used to read the FILTRANE specfile in order to possibly modify the algoritmic default parameters (see Section 2.7.1).
3. The subroutine FILTRANE_solve is called to solve the problem by applying the FILTRANE algorithm.
4. The subroutine FILTRANE_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by FILTRANE, at the end of the solution process. It is important to do this if the data object is re-used for another problem with a different structure since FILTRANE_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

2.4.1 The initialization subroutine

Default values for the control parameters are provided as follows:

```fortran
CALL FILTRANE_initialize( control, inform, data )
```

control is a scalar INTENT(OUT) argument of type FILTRANE_control_type (see Section 2.3.2). On exit, control contains default values for the components as described in Section 2.3.2. These values should only be changed after calling FILTRANE_initialize.

call inform is a scalar INTENT(OUT) argument of type FILTRANE_inform_type (see Section 2.3.3). A successful call to the routine FILTRANE_initialize is indicated when the component status has the value 0. For other return values of status, see Sections 2.5 and 2.6.

data is a scalar INTENT(OUT) argument of type FILTRANE_data_type (see Section 2.3.4). It is used to hold data about the problem being solved. It should never be altered by the user, except for returning values to FILTRANE_solve via the reverse communication interface (see Section 2.5).

2.4.2 The subroutine that applies the FILTRANE algorithm to the problem

The FILTRANE solver is called as follows:

```fortran
CALL FILTRANE_solve( problem, control, inform, data )
```

Such a call must always be preceded by a call to FILTRANE_initialize.

problem is a scalar INTENT(OUT) argument of type NLPT_problem_type that contains the problem statement.

On input, its n, m, x, xmin, xstatus, c, cu, cl, csize, Jtype and infinity components must be set. In addition, its c, y, g, equation, Jval, J_row and J_col components must be allocated (see Section 2.3.1).

On successful output, the following components of the problem data type are of interest:

x now contains the values of the variables at the point where FILTRANE was terminated,
f contains the value of the objective function $\theta$ at the point $x$.
g contains the gradient of $\theta$ at the point $x$.

If \( \text{problem}%m > 0 \), then

c contains the values of the constraints at the point $x$.

If, additionally, \( \text{control}%\text{external}_\text{Jacobian}\text{-products} \) is .FALSE., then

\( J_{\text{val}} \) contains the values of the nonzero entries of the constraints Jacobian at $x$,
\( J_{\text{row}} \) contains the row indices of the the nonzero entries of the constraints Jacobian at $x$,
\( J_{\text{col}} \) contains the column indices of the the nonzero entries of the constraints Jacobian at $x$.

\( \text{control} \) is a scalar \( \text{INTENT(IN)} \) argument of type \( \text{FILTRANE}_\text{control}_\text{type} \) (see Section 2.3.2). Default values may be assigned by calling \( \text{FILTRANE}_\text{initialize} \) prior to the first call to \( \text{FILTRANE}_\text{solve} \).

\( \text{info} \) is a scalar \( \text{INTENT(OUT)} \) argument of type \( \text{FILTRANE}_\text{inform}_\text{type} \) (see Section 2.3.3). A successful call to the routine \( \text{FILTRANE}_\text{solve} \) is indicated when the component \( \text{status} \) has the value 0. For other return values of \( \text{status} \), see Section 2.6.

\( \text{data} \) is a scalar \( \text{INTENT(INOUT)} \) argument of type \( \text{FILTRANE}_\text{data}_\text{type} \) (see Section 2.3.4). It is used to hold data about the problem being solved. It must never be altered by the user since the last call to any of the \( \text{FILTRANE} \) routines, except for returning values to \( \text{FILTRANE}_\text{solve} \) via the reverse communication interface (see Section 2.5).

2.4.3 The termination subroutine

All previously allocated workspace arrays for \( \text{FILTRANE} \) are deallocated as follows:

\[ \text{CALL FILTRANE}_\text{terminate}( \text{control}, \text{info}, \text{data} ) \]

\( \text{control} \) is a scalar \( \text{INTENT(IN)} \) argument of type \( \text{FILTRANE}_\text{control}_\text{type} \) exactly as for \( \text{FILTRANE}_\text{initialize} \).

\( \text{info} \) is a scalar \( \text{INTENT(OUT)} \) argument of type \( \text{FILTRANE}_\text{inform}_\text{type} \) exactly as for \( \text{FILTRANE}_\text{initialize} \). A successful call to \( \text{FILTRANE}_\text{terminate} \) is indicated when the component \( \text{status} \) has the value 0. For other return values of \( \text{status} \), see Section 2.6.

\( \text{data} \) is a scalar \( \text{INTENT(INOUT)} \) argument of type \( \text{FILTRANE}_\text{data}_\text{type} \) exactly as for \( \text{FILTRANE}_\text{solve} \), which must not have been altered by the user since the last call to \( \text{FILTRANE}_\text{initialize} \), except for returning values to \( \text{FILTRANE}_\text{solve} \) via the reverse communication interface (see Section 2.5). On exit, array components will have been deallocated.

Note that a call to this routine is mandatory before \( \text{FILTRANE}_\text{solve} \) is called for a new problem whose structure differs from the current one.

2.5 Reverse communication

A positive value of \( \text{info}\%\text{status} \) on exit from \( \text{FILTRANE}_\text{solve} \) indicates that the user needs to take appropriate action before re-entering the subroutine. Possible values are:

1. The user must compute, at the point given in the \( \text{problem}\%x \),
   - the values of the constraints,
   - and place the result in \( \text{problem}\%c(1:\text{problem}%m) \),
• the values of the nonzero entries of the constraints Jacobian, and place the result in problem\%J_val(1:problem\%J_size),
• the row indices of the nonzero entries of the constraints Jacobian, and place the result in problem\%J_row(1:problem\%J_size),
• the column indices of the nonzero entries of the constraints Jacobian, and place the result in problem\%J_col(1:problem\%J_size).

No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again. This is only used at the initial (starting) point.

2. The user must compute, at the point given in the problem\%x,
• the values of the constraints, and place the result in problem\%c(1:problem\%m),
• the values of the nonzero entries of the constraints Jacobian, and place the result in problem\%J_val(1:problem\%J_size).

No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again. This is only possibly used at the exit of FILTRANE when control\%save\_best\_point is .TRUE..

3, 4 and 5. The user must compute the values of the constraints at the point given in the problem\%x, and place the result in problem\%c(1:problem\%m). No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again.

6. The user must compute the values of the nonzero entries of the constraints Jacobian at the point given in the problem\%x, and place the result in problem\%J_val(1:problem\%J_size). No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again.

7. The user must compute the product \( J(x)v \), where \( v \) is given by data\%RC\_v(1:problem\%n) and place the result in the vector data\%RC\_Mv(1:problem\%m). No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again. This only occurs if control\%external\_J\_products is .TRUE. (see Section 2.3.2).

8, 9, 10 and 11. The user must compute the product \( J(x)^Tv \), where \( v \) is given by data\%RC\_v(1:problem\%m) and place the result in the vector data\%RC\_Mv(1:problem\%m). No other argument of FILTRANE\_solve may be modified before FILTRANE\_solve is called again. This only occurs if control\%external\_J\_products is .TRUE. (see Section 2.3.2).

12, 13 and 14. The user must apply a preconditioner for the model’s Hessian to data\%RC\_Pv(1:problem\%n) and place the result in the same vector. The model’s Hessian is given by

\[
\frac{\alpha(\alpha - 1)}{2} \left\{ \begin{array}{c} J(x) \text{diag} \left( \left[ \theta_c(x) \right]^{\alpha-2}_i \right) J(x)^T + \text{diag} \left( \left[ \theta_c(x) \right]^{\alpha-2}_i \right) \end{array} \right\} + \frac{\alpha}{2} \sum_{i=1}^{m} \left| \theta_c(x) \right|^{\alpha-1}_i \psi_{\chi}^2 \, \chi_i(x),
\]

where \( \theta_c(x) \) contains the first \( m \) components of \( \theta(x) \) (the constraints’ violations) and \( \theta_b(x) \) the last \( n \) (the bounds’ violations), where \( \alpha \) is the exponent of the inequality penalty function used (by default: \( \alpha = 2 \) for the Euclidean norm, see the information on control\%inequality\_penalty\_type in Section 2.3.2), and where the last term is only present if the full Newton model is used at the current iteration. This last condition can be checked by verifying that the value of data\%model\_used is equal to NEWTON. If data\%model\_used is equal to GAUSS\_NEWTON instead, then the last term does not appear in the expression of the Hessian. No other argument of FILTRANE\_solve than data\%RC\_Pv may be modified before FILTRANE\_solve is called again. This only occurs if control\%prec\_used is USER\_DEFINED (see Section 2.3.2).
15 and 16. The user must compute the product

\[ \sum_{i=1}^{m} y_i \nabla c_i(x) v, \]

where

- \( y \) is given by problem%y(1:problem%m),
- \( x \) is given by problem%x(1:problem%n),
- \( v \) is given by data%RCv(1:problem%n),

and place the result in the vector data%RCMv(1:problem%n). No other argument of FILTRANE_solve may be modified before FILTRANE_solve is called again. The user may use the fact that data%RCnewx is .TRUE. iff the current product request is the first that involves the same vector problem%x. This only occurs if control%model_type is NEWTON or ADAPTIVE (see Section 2.3.2).

### 2.6 Warning and error messages

A negative value of info%status on exit from FILTRANE_initialize, FILTRANE_read_specfile, FILTRANE_apply, FILTRANE_restore, or FILTRANE_terminate indicates that an error has occurred. No further calls should be made to the four three of these routines until the error has been corrected. Possible values are:

-1. The memory allocation failed.
-2. A file intended for saving checkpointing information could not be opened.
-3. An IO error occurred while saving checkpointing information on the relevant disk file.
-5. Further progress of the algorithms appears to be impossible, although successful termination is not recognized. This may happen if the problem is extremely ill-conditioned, if the current preconditioner is inefficient, or if there are errors in the calculation of the constraints Jacobian. In the first case, it may happen that the current iterate gives a reasonable approximation of the solution (the components of the problem data type can then be interpreted as for successful termination).
-6. The maximum number of iterations has been reached and computation terminated.
-8. The number of variables is non-positive.
-9. The number of constraints is negative.
-21. The information contained in the checkpointing file could not be read or does not correspond to the problem being solved.
-22. The step could not be computed by the GALAHAD_GLTR procedure.
-23. The dimension of the gradient problem%g is not equal to the number of variables in the problem problem%n.
-24. One of the vectors problem%x, problem%x_l, problem%x_u, problem%c, problem%c_l, problem%c_u, problem%y, problem%g, problem%J_val, problem%J_col, problem%J_row, problem%equation or problem%x_status is not allocated on input, although it should be.
-25. The user-supplied number of groups is either negative or exceeds the number of constraints plus the number of bounded variables.
-26. The vector control%group is not associated although user-defined groups are requested.

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.
-27. The dimension of the vector control%group is different from the sum of the number of constraints and the number of bounded variables.

-28. The user-supplied group index (for a constraint or a bound) is either negative, or exceeds control%nbr_groups.

-29. FILTRANE was re-entered (in the reverse communication protocol) with an invalid value for inform%status.

-100. This should not happen! (If it does anyway, please report (with problem data and specfile) to Ph. Toint. Thanks in advance.)

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 FILTRANE_control_type (see Section 2.3.2), by reading an appropriate data specification file using the subroutine FILTRANE_read_specfile. This facility is useful as it allows a user to change FILTRANE control parameters without editing and recompiling programs that call FILTRANE.

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

```
BEGIN FILTRANE
  keyword  value
  ........  ....
  keyword  value
END
```

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

and

```
END FILTRANE SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN FILTRANE" 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 off" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameter may be of five different types, namely integer, logical, real, string or symbol. 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", "F.FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

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.
String are specified as a sequence of characters. A symbolic value is a special string obtained from one of the predefined symbols of the SYMBOLS module by deleting the leading `GALAHAD` characters in its name. Thus, the specification command

```
print-level SILENT
```

implies that the value `GALAHAD_SILENT` is assigned to `control%print_level`. This technique is intended to help expressing an (integer) control parameter for an algorithm in a “language” that is close to natural (see Section 2.2).

The specification file must be open for input when `FILTRANE_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 `FILTRANE_read_specfile`.

### 2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL FILTRANE_read_specfile( device, control, inform )
```

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`.

control is a scalar INTENT(INOUT) argument of type `FILTRANE_control_type` (see Section 2.3.2). Default values should have already been set, perhaps by calling `FILTRANE_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 2.3.2) of control that each affects are given in Tables 2.1–2.2.

inform is a scalar INTENT(OUT) argument of type `FILTRANE_inform_type` (see Section 2.3.3).

<table>
<thead>
<tr>
<th>command</th>
<th>component of control</th>
<th>value type/symbolic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>printout-device</td>
<td>%out</td>
<td>integer</td>
</tr>
<tr>
<td>error-printout-device</td>
<td>%errout</td>
<td>SILENT, TRACE, ACTION, DETAILS, DEBUG, CRAZY</td>
</tr>
<tr>
<td>print-level</td>
<td>%print_level</td>
<td>integer, SILENT, TRACE, ACTION, DETAILS, DEBUG, CRAZY</td>
</tr>
<tr>
<td>start-printing-at-iteration</td>
<td>%start_print</td>
<td>integer</td>
</tr>
<tr>
<td>stop-printing-at-iteration</td>
<td>%stop_print</td>
<td>integer</td>
</tr>
<tr>
<td>residual-accuracy</td>
<td>%c_accuracy</td>
<td>real</td>
</tr>
<tr>
<td>gradient-accuracy</td>
<td>%g_accuracy</td>
<td>real</td>
</tr>
<tr>
<td>stop-on-preconditioned-gradient-norm</td>
<td>%stop_on_prec_g</td>
<td>logical</td>
</tr>
<tr>
<td>stop-on-maximum-gradient-norm</td>
<td>%stop_on_g_max</td>
<td>logical</td>
</tr>
<tr>
<td>maximum-number-of-iterations</td>
<td>%max_iterations</td>
<td>logical</td>
</tr>
</tbody>
</table>

Table 2.1: Specfile commands and associated components of control (part 1).
<table>
<thead>
<tr>
<th>command</th>
<th>component of control</th>
<th>value type/symbolic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>model-type</td>
<td>%model_type</td>
<td>GAUSS_NEWTON, NEWTON, ADAPTIVE</td>
</tr>
<tr>
<td>automatic-model-inertia</td>
<td>%model_inertia</td>
<td>integer</td>
</tr>
<tr>
<td>automatic-model-criterion</td>
<td>%model_criterion</td>
<td>BEST_FIT, BEST_REDUCTION</td>
</tr>
<tr>
<td>maximum-number-of-cg-iterations</td>
<td>%max_cg_iterations</td>
<td>logical</td>
</tr>
<tr>
<td>subproblem-accuracy</td>
<td>%subproblem_accuracy</td>
<td>ADAPTIVE, FULL</td>
</tr>
<tr>
<td>relative-subproblem-accuracy-power</td>
<td>%glt_accuracy_power</td>
<td>real</td>
</tr>
<tr>
<td>minimum-relative-subproblem-accuracy</td>
<td>%min_glt_accuracy</td>
<td>real</td>
</tr>
<tr>
<td>preconditioner-used</td>
<td>%prec_used</td>
<td>NONE, BANDED</td>
</tr>
<tr>
<td>semi-bandwidth-for-band-preconditioner</td>
<td>%semi_bandwidth</td>
<td>integer</td>
</tr>
<tr>
<td>external_Jacobian_products</td>
<td></td>
<td>logical</td>
</tr>
<tr>
<td>equations-grouping</td>
<td>%grouping</td>
<td>NONE, AUTOMATIC, USER_DEFINED</td>
</tr>
<tr>
<td>number-of-groups</td>
<td>%nbr_groups</td>
<td>integer</td>
</tr>
<tr>
<td>balance-initial-group-values</td>
<td>%balance_group_values</td>
<td>logical, NEVER, INITIAL, ALWAYS</td>
</tr>
<tr>
<td>use-filter</td>
<td>%use_filter</td>
<td>integer</td>
</tr>
<tr>
<td>maximum-filter-size</td>
<td>%maximal_filter_size</td>
<td>integer</td>
</tr>
<tr>
<td>filter-size-increment</td>
<td>%filter_size_increment</td>
<td>integer</td>
</tr>
<tr>
<td>filter-margin-type</td>
<td>%filter_margin_type</td>
<td>CURRENT, FIXED, SMALLEST</td>
</tr>
<tr>
<td>filter-margin-factor</td>
<td>%gamma_f</td>
<td>real</td>
</tr>
<tr>
<td>remove-dominated-entries</td>
<td>%remove-dominated</td>
<td>logical</td>
</tr>
<tr>
<td>weak-acceptance-power</td>
<td>%weak_accept_power</td>
<td>real</td>
</tr>
<tr>
<td>minimum-weak-acceptance-factor</td>
<td>%min_weak_accept_factor</td>
<td>real</td>
</tr>
<tr>
<td>initial-radius</td>
<td>%initial_radius</td>
<td>real</td>
</tr>
<tr>
<td>initial-TR-relaxation-factor</td>
<td>%itr_relax</td>
<td>real</td>
</tr>
<tr>
<td>secondary-TR-relaxation-factor</td>
<td>%str_relax</td>
<td>real</td>
</tr>
<tr>
<td>minimum-rho-for-successful-iteration</td>
<td>%eta_1</td>
<td>real</td>
</tr>
<tr>
<td>minimum-rho-for-very-successful-iteration</td>
<td>%eta_2</td>
<td>real</td>
</tr>
<tr>
<td>radius-increase-factor</td>
<td>%gamma_2</td>
<td>real</td>
</tr>
<tr>
<td>radius-reduction-factor</td>
<td>%gamma_1</td>
<td>real</td>
</tr>
<tr>
<td>worst-case-radius-reduction-factor</td>
<td>%gamma_0</td>
<td>real</td>
</tr>
<tr>
<td>save-best-point</td>
<td>%save_best_point</td>
<td>logical</td>
</tr>
<tr>
<td>checkpointing-frequency</td>
<td>%checkpoint_freq</td>
<td>integer</td>
</tr>
<tr>
<td>checkpointing-device</td>
<td>%checkpoint_dev</td>
<td>integer</td>
</tr>
<tr>
<td>checkpointing-file</td>
<td>%checkpoint_file</td>
<td>character(30)</td>
</tr>
<tr>
<td>restart-from-checkpoint</td>
<td>%restart_from_checkpoint</td>
<td>logical</td>
</tr>
</tbody>
</table>

Table 2.2: Specfile commands and associated components of control (part 2).
2.8 Information printed

The meaning of the various control\{print\}_level values is defined as follows:

**GALAHAD\_SILENT**: no printout is produced,

**GALAHAD\_TRACE**: only reports a one line summary of each iteration. This summary includes the current values of the objective function, the (possibly preconditioned) norm of its gradient, the ratio $\rho$ of achieved to predicted reduction, the norm of the step and the trust-region radius. It also reports the cumulative number of GLT iterations, the iteration type and the number of entries currently in the filter.

The iteration type is a four character string whose interpretation requires some detailed knowledge of the algorithm (see Section 4 and the references therein). The first character describes the model type used at the current iteration:

- **G**: the Gauss-Newton model was used,
- **N**: the full Newton model was used.

The second character describes the type of step that was allowed:

- **R**: the current step was restricted to lie in the trust region,
- **U**: the current step was not restricted to lie in the trust region.

The third character described the manner in which the **GALAHAD\_GLTR** procedure has been terminated for the step computation:

- **I**: the stopping criterion was met for a step internal to the trust-region,
- **B**: the stopping criterion was met for a step lying on the trust-region boundary,
- **E**: the stopping criterion was met for a step exterior to the trust region,
- **M**: the maximum number of iterations allowed for **GALAHAD\_GLTR** has been reached.

The fourth character describes the result of applying the various acceptance tests to the trial point:

- **W**: the trial point was acceptable for the weak acceptance test (2.2),
- **F**: the trial point was acceptable for the filter, but its violation was not included in the filter,
- **f**: the trial point was acceptable for the filter, and its violation was included in the filter,
- **S**: the trial point was accepted as very successful by the trust-region tests,
- **s**: the trial point was accepted as successful by the trust-region tests,
- **u**: the trial point was rejected as unsuccessful by the trust-region tests,
- **U**: the trial point was rejected as very unsuccessful by the trust-region tests.

**GALAHAD\_ACTION**: additionally reports the mains steps of each iteration,

**GALAHAD\_DETAILS**: additionally reports the values of variables, constraints, Jacobian entries, gradient and step components at each iteration,

**GALAHAD\_DEBUG**: additionally reports LOTS of information, including details of subprocesses within each iteration,

**GALAHAD\_CRAZY**: reports a completely silly amount of information.

*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](http://galahad.rl.ac.uk/galahad-www/cou.html) for full details.*
3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: FILTRANE_solve calls the BLAS functions *NRM2, *DOT and *SWAP, where * is S for the default real version and D for the double precision version.

Other modules used directly: FILTRANE calls the GALAHAD modules NLPT, GLTR, BAND, SYMBOLS, SORT, TOOLS and SPECFILE.

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

Restrictions: problem%n > 0, problem%m ≥ 0.

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

4 METHOD

The method used by FILTRANE is iterative and combines multidimensional filter and trust-region techniques. Trust-region methods build a local (in the case of FILTRANE, quadratic) approximation of \( \theta(x) \) and then compute a step that decreases the value of that model in a “trust region”, where the model is deemed to approximate the true function well enough. The new trial point is then compared to previous iterates. If it provides reduction in the violation of at least one of the (groups of) constraints, it is accepted as the new iterate, according to the multidimensional filter acceptance criterion, or if it provides sufficient reduction in the objective function. If the trial point is rejected, the trust-region radius is reduced and another step computed in the smaller region, until a trial point can be accepted.

The calculation of the step is performed using the Generalized Lanczos Trust Region Method of Gould, Lucidi and Toint, as implemented in the GALAHAD GLTR module.

The package has a full reverse-communication interface.

The use of FILTRANE for the solution of a problem follows one on the two sequences:

1. initialize \( \rightarrow \) [ solve \( \rightarrow \) terminate ]
2. initialize \( \rightarrow \) [ read specfile \( \rightarrow \) solve \( \rightarrow \) terminate ]

where the procedure’s control parameter may be modified by reading the specfile (see Section 2.7). Each of the “boxed” steps in these sequences corresponds to calling a specific routine of the package (see Section 2.4). In the above diagrams, brackated subsequence of steps means that they can be repeated.

References:

The algorithm is described in more detail in


N. Gould and Ph. L. Toint (2003), FILTRANE, a Fortran 95 + TR 15581 or Fortran 2003 filter-trust-region package for solving nonlinear feasibility problems, Technical report 03/??, The University of Namur, Belgium.

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.
### 5 Example of Use

Suppose that we wish to apply FILTRANE to solve the nonlinear feasibility problem for the constraints

\[
3x_1^2 + 2x_2^3 + x_1x_2 = 0, \\
x_1 + x_2 = 0, \\
\begin{align*}
-2 &\leq x_1 \leq 2 \quad \text{and} \quad -2 \leq x_2 \leq 2,
\end{align*}
\]

starting from the initial point \(x^T = (11)\). We thus have that \(n = 2, m = 2\). Computing the constraints Jacobian, we verify that it is given by

\[
J(x) = \begin{pmatrix}
6x_1 + x_2 & x_1 + 6x_2^2 \\
1 & 1
\end{pmatrix}.
\]

We may then use the following code:

```fortran
! THIS VERSION: GALAHAD 2.1 - 13/02/2008 AT 09:40 GMT.
PROGRAM GALAHAD_FILTRANE_EXAMPLE
USE GALAHAD_NLPT_double ! the problem type
USE GALAHAD_FILTRANE_double ! the FILTRANE solver
USE GALAHAD_SYMBOLS
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
INTEGER, PARAMETER :: ispec = 55 ! SPECfile device number
INTEGER, PARAMETER :: iout = 6 ! stdout and stderr
REAL( KIND = wp ), PARAMETER :: INFINITY = (10.0_wp)**19
TYPE( NLPT_problem_type ) :: problem
TYPE( FILTRANE_control_type ) :: FILTRANE_control
TYPE( FILTRANE_inform_type ) :: FILTRANE_inform
TYPE( FILTRANE_data_type ) :: FILTRANE_data
INTEGER :: J_size
REAL( KIND = wp ), DIMENSION( 3 ) :: H1
! Set the problem up.
problem%n = 2
ALLOCATE( problem%x( problem%n ) , problem%x_status( problem%n ), &
\problem%x_l( problem%n ), problem%x_u( problem%n ), &
\problem%g( problem%n ) , problem%z( problem%n ) )
problem%m = 2
ALLOCATE( problem%equation( problem%m ), &
\problem%c( problem%m ) , problem%c_l( problem%m ), &
\problem%c_u( problem%m ), problem%y( problem%m ) )
problem%J_ne = 4
J_size = problem%J_ne + problem%n
ALLOCATE( problem%J_val( J_size), problem%J_row( J_size), &
\problem%J_col( J_size ) )
problem%J_type = GALAHAD_COORDINATE
problem%infinity = INFINITY
problem%x = (/ 1.0D0, 1.0D0 /)
problem%x_l = (/ -2.0D0, -2.0D0 /)
problem%x_u = (/ 2.0D0, 2.0D0 /)
problem%c_l = (/ 0.0D0, 0.0D0 /)
problem%c_u = (/ 0.0D0, 0.0D0 /)
problem%equation = (/ .TRUE., .TRUE. /)
! Initialize FILTRANE.
CALL FILTRANE_initialize( FILTRANE_control, FILTRANE_inform, FILTRANE_data )
```

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](http://galahad.rl.ac.uk/galahad-www/cou.html) for full details.
! Read the FILTRANE spec file (not necessary in this example, as the default
! settings are mostly suitable).
! OPEN( ispec, file = 'FILTRANE.SPC', form = 'FORMATTED', status = 'OLD' )
! CALL FILTRANE_read_specfile( ispec, FILTRANE_control, FILTRANE_inform )
! CLOSE( ispec )
! Nevertheless... ask for some output:
FILTRANE_control%print_level = GALAHAD_TRACE
! Now apply the solver in the reverse communication loop.
DO
   CALL FILTRANE_solve( problem, FILTRANE_control, FILTRANE_inform, &
FILTRANE_data )
   SELECT CASE ( FILTRANE_inform%status )
   CASE ( 1, 2 ) ! constraints values and Jacobian
      problem%c( 1 ) = 3.0D0 * problem%x( 1 ) ** 2 +
      2.0D0 * problem%x( 2 ) ** 3 +
      problem%x( 1 ) * problem%x( 2 )
      problem%c( 2 ) = problem%x( 1 ) + problem%x( 2 )
      problem%J_val( 1 ) = 6.0D0 * problem%x( 1 ) + problem%x( 2 )
      problem%J_val( 2 ) = 1.0D0
      problem%J_row( 1 : 4 ) = (/ 1, 2, 1, 2 /)
      problem%J_col( 1 : 4 ) = (/ 1, 1, 2, 2 /)
   CASE ( 3 : 5 ) ! constraints values only
      problem%c( 1 ) = 3.0D0 * problem%x( 1 ) ** 2 +
      2.0D0 * problem%x( 2 ) ** 3 +
      problem%x( 1 ) * problem%x( 2 )
      problem%c( 2 ) = problem%x( 1 ) + problem%x( 2 )
   CASE ( 6 ) ! Jacobian only
      problem%J_val( 1 ) = 6.0D0 * problem%x( 1 ) + problem%x( 2 )
      problem%J_val( 2 ) = 1.0D0
      problem%J_val( 3 ) = 6.0D0 * problem%x( 2 ) ** 2 + problem%x( 1 )
      problem%J_val( 4 ) = 1.0D0
   CASE ( 15, 16 ) ! product times the Hessian of the Lagrangian
      IF ( FILTRANE_data%RC_newx ) THEN
         H1( 1 ) = 6.0D0
         H1( 2 ) = 1.0D0
         H1( 3 ) = 12.0D0 * problem%x( 2 )
      END IF
      FILTRANE_data%RC_Mv( 1 ) =
      problem%y( 1 ) * H1( 1 ) * FILTRANE_data%RC_v( 1 ) +
      problem%y( 2 ) * H1( 2 ) * FILTRANE_data%RC_v( 2 )
   CASE DEFAULT
      EXIT
   END SELECT
END DO ! end of the reverse communication loop
! Terminate FILTRANE.
FILTRANE_control%print_level = GALAHAD_SILENT
CALL FILTRANE_terminate( FILTRANE_control, FILTRANE_inform, FILTRANE_data )
! Output results.
This produces the following output:

```
Iter  f(x)  ||g(x)||  rho  ||s||  Delta  #CGits  Type  F
0  2.000E+01  6.223E+01  1.000E+00  0  0
1  1.916E+01  8.965E+00  9.341E+10  6.223E+01  1.000E+00  1 GUIF 0
2  6.430E+00  1.538E+01  6.644E+00  1.245E+00  1.000E+00  3 GUIF 1
3  7.325E-01  4.115E-01  9.986E-01  8.279E-02  1.245E+00  4 GUIF 1
4  2.932E-03  6.961E-01  5.997E+01  3.929E-01  1.245E+00  6 GUIF 1
5  6.746E-07  1.380E-02  9.999E-01  8.239E-03  1.245E+00  7 GUIF 1
6  8.208E-07  1.103E-02  9.878E-01  4.879E-02  1.245E+00  9 GUIF 1
```

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](http://galahad.rl.ac.uk/galahad-www/cou.html) for full details.
Problem successfully solved: constraints violations are small.

Problem : GALEXAMPLE

X1 = 1.0000000219305E+00
X2 = -1.0000000219305E+00

C1 = -4.386100604936E-07
C2 = 0.000000000000E+00

Exit condition number = 0
Objective function value = 9.618939258311E-14
Number of iterations = 8
Number of CG iterations = 12
Number of constraints evaluations = 9
Number of Jacobian evaluations = 9

We could also make use of the FILTRANE_read_specfile routine to set the printing level, in which case the lines are replaced by

! Nevertheless... ask for some output:
   FILTRANE_control$print_level = GALAHAD_TRACE

where we assume that the file FILTRANE.SPC exists in the current directory and contains the lines

BEGIN FILTRANE SPECIFICATION
   print-level TRACE
END FILTRANE SPECIFICATION