GALAHAD FILTRANE

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

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

$$\mathbf{c}^l \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}^u,$$

and

$$\mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u,$$

where $\mathbf{c}(x)$ is a smooth function from $\mathbb{R}^n$ into $\mathbb{R}^m$ and where inequalities are understood componentwise. The vectors $\mathbf{c}^l \leq \mathbf{c}^u$ and $\mathbf{x}^l \leq \mathbf{x}^u$ 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 $\mathbf{c}^l$ or $\mathbf{c}^u$ is finite, while, for j between 1 and n, the j-th components of both $\mathbf{x}^l$ and $\mathbf{x}^u$ are allowed to be infinite. In what follows, we will say that variable j is bounded if the j-th component of either $\mathbf{x}^l$ or $\mathbf{x}^u$ is finite. Equalities may be specified by choosing identical lower and upper bounds on either $\mathbf{c}$ or $\mathbf{x}$. 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

$$\min_x \tfrac{1}{2}\|\theta(\mathbf{x})\|^2, \tag{1.1}$$

where we define

$$\theta(x) \stackrel{\text{def}}{=} \begin{pmatrix} \max([\mathbf{c}^l - \mathbf{c}(\mathbf{x})]_+, [\mathbf{c}(\mathbf{x}) - \mathbf{c}^u]_+) \\ \max([\mathbf{x}^l - \mathbf{x}]_+, [\mathbf{x} - \mathbf{x}^u]_+) \end{pmatrix} \in \mathbb{R}^p,$$

where, for a vector $\mathbf{y}$, $[\mathbf{y}]_+ = \max[0, \mathbf{y}]$, and where all maxima are taken componentwise.

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

$$\theta_\ell(x) \stackrel{\text{def}}{=} \left\| \begin{array}{c} \max([\mathbf{c}^l_{[\ell]} - \mathbf{c}_{[\ell]}(\mathbf{x})]_+, [\mathbf{c}_{[\ell]}(\mathbf{x}) - \mathbf{c}^u_{[\ell]}]_+) \\ \max([\mathbf{x}^l_{[\ell]} - \mathbf{x}_{[\ell]}]_+, [\mathbf{x}_{[\ell]}]_+) - \mathbf{x}^u_{[\ell]} \end{array} \right\| \quad \text{for } \ell = 1, \ldots, p,$$

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

ATTRIBUTES — Versions: GALAHAD_FILTRANE_single, GALAHAD_FILTRANE_double.

Uses: GALAHAD_NLPT, GALAHAD_SPECFILE, GALAHAD_GLTR, GALAHAD_BAND, GALAHAD_SYMBOLS, GALAHAD_TOOLS, *NRM2, *DOT, *SWAP.

Date: May 2003. **Origin:** Ph. L. Toint, The University of Namur, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

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

```
USE GALAHAD_FILTRANE_single
```

with the obvious substitution GALAHAD_FILTRANE_double, GALAHAD_FILTRANE_single_64 and GALAHAD_FILTRANE_double_64 for the other variants.

If it is required to use more than one of the modules at the same time, the derived types NLPT_problem_type, FILTRANE_control_type, FILTRANE_inform_type and FILTRANE_data_type (Section 2.4) and the four subroutines FILTRANE_initialize, FILTRANE_read_specfile, FILTRANE_solve, FILTRANE_terminate, (Section 2.5) must be renamed on one of the USE statements.

2.1 Matrix storage formats

The constraint Jacobian matrix $\mathbf{J}(\mathbf{x})$ 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 $\mathbf{J}(\mathbf{x})$, its row index i, column index j and value $\mathbf{J}_{ij}$ 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 Real and integer kinds

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

2.3 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 clarify 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.

2.4 The derived data types

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

2.4.1 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 INTEGER(ip_), that holds the number of problem variables, n.

m is a scalar variable of type INTEGER(ip_), that holds the number of problem constraints, m.

x is a rank-one allocatable array of dimension n and type REAL(rp_), 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 REAL(rp_), that holds the vector of lower bounds $\mathbf{x}^l$ on the problem variables. The j-th component of x_l, $j = 1,\ldots,n$, contains $\mathbf{x}^l_j$. Infinite bounds are allowed by setting the corresponding components of x_l to any value smaller than that -infinity.

x_u is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the vector of upper bounds $\mathbf{x}^u$ on the problem variables. The j-th component of x_u, $j = 1,\ldots,n$, contains $\mathbf{x}^u_j$. Infinite bounds are allowed by setting the corresponding components of x_u to any value larger than that infinity.

c_l is a rank-one allocatable array of dimension m and type REAL(rp_), that holds the vector of lower bounds $\mathbf{c}^l$ on the general constraints. The i-th component of c_l, $i = 1,\ldots,m$, contains $\mathbf{c}^l_i$. 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 REAL(rp_), that holds the vector of upper bounds $\mathbf{c}^l$ on the general constraints. The i-th component of c_u, $i = 1,\ldots,m$, contains $\mathbf{c}^u_i$. 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 INTEGER(ip_), which holds the number of nonzero entries of the constraints Jacobian matrix.

J_type is a scalar variable of type INTEGER(ip_), 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 REAL(rp_), 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 INTEGER(ip_),

c is a rank-one allocatable array of dimension m and type REAL(rp_),

y is a rank-one allocatable array of dimension m and type REAL(rp_),

g is a rank-one allocatable array of dimension n and type REAL(rp_),

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 REAL(rp_),

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

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

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 2.4.2), in which case the values of J_ne and J_type are again irrelevant.

2.4.2 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 FILTRANE_initialize (see Section 2.5.1), while individual components may also be changed by calling FILTRANE_read_specfile (see Section 2.8.1). The components of FILTRANE_control_type are:

c_accuracy is a scalar variable of type REAL(rp_), 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 REAL(rp_), that specifies an accuracy threshold such that the FILTRANE iteration is successfully terminated if the (possibly preconditioned) Euclidean norm of $\nabla_x \theta$ is under the threshold. The default is g_accuracy = 10^{-4} in single precision, and c_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_bandwdith = 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 INTEGER(ip_), 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 INTEGER(ip_), 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 INTEGER(ip_), 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 INTEGER(ip_), 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 INTEGER(ip_) 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 ℓ-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 `INTEGER(ip_)`, 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 2.6).

The default is `prec_used` = GALAHAD_NONE.

`semi_bandwidth` is a scalar variable of type `INTEGER(ip_)`, 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 precondtioning) 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 2.6), in which case the Jacobian is not used at all inside `FILTRANE`.

`out` is a scalar variable of type `INTEGER(ip_)`, 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 `INTEGER(ip_)`, that holds the unit number associated with the device used for error ouput. The default is `errout` = 6.

`print_level` is a scalar variable of type `INTEGER(ip_)`, that holds the level of printout requested by the user. See Section 2.9. The default is `print_level` = GALAHAD_SILENT.

`start_print` is a scalar variable of type `INTEGER(ip_)`, 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 `INTEGER(ip_)`, 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 `INTEGER(ip_)`, 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 `INTEGER(ip_)`, 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 `INTEGER(ip_)`, 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 `INTEGER(ip_)`, that specifies the type of penalty function used to measure constraint violations. Valid values are:

2: the ℓ_2 penalty function is used,

3: the ℓ_3 penalty function is used,

4: the ℓ_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 `INTEGER(ip_)`, 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

$$\min\left[\varepsilon_1, \|r_0\|^{\varepsilon_2}\right].\|r_0\| \tag{2.1}$$

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 `REAL(rp_)`, that holds the minimum relative accuracy ε_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 `REAL(rp_)`, that holds the power ε_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 `INTEGER(ip_)`, 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 `INTEGER(ip_)`, 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 vale 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 `INTEGER(ip_)`, 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 `REAL(rp_)`, 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 `REAL(rp_)`, 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^{20}.

`str_relax` is a scalar variable of type `REAL(rp_)`, 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 `REAL(rp_)`, that holds the power α_2 in the "weak acceptance criterion" that accepts a trial point x_k^+ if

$$\theta(x_k) - \theta(x_k^+) \geq \alpha_1 \min\left[1, \theta(x_k)^{\alpha_2}\right]. \tag{2.2}$$

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 `REAL(rp_)`, that holds the parameter α_1 in (2.2). It is only relevant if `weak_accept_power` ≥ 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 `REAL(rp_)`, that holds the initial trust-region radius. It si 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 `REAL(rp_)`, 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 `REAL(rp_)`, 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 REAL(rp_), 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 REAL(rp_), 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 REAL(rp_), 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 INTEGER(ip_), 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 CHARACTERof length 30, that holds the name of the file use for storing checkpointing information on disk. The default is checkpoint_file = FILTRANE.sav.

checkpoint_dev is a scalar variable of type INTEGER(ip_), 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..

2.4.3 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 INTEGER(ip_), that gives the exit status of the algorithm. See Sections 2.7 and 2.9 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 INTEGER(ip_), that gives the the final number of FILTRANE iterations.

nbr_cg_iterations is a scalar variable of type INTEGER(ip_), 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 INTEGER(ip_), that gives the final number of evaluations of the constraints values.

nbr_J_evaluations is a scalar variable of type INTEGER(ip_), that gives the final number of Jacobian evaluations.

2.4.4 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 2.6).

2.5 Argument lists and calling sequences

There are four procedures for user calls (see Section 2.8 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 2.8.1).
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.

2.5.1 The initialization subroutine

Default values for the control parameters are provided as follows:

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

`control` is a scalar `INTENT(OUT)` argument of type `FILTRANE_control_type` (see Section 2.4.2). On exit, `control` contains default values for the components as described in Section 2.4.2. These values should only be changed after calling `FILTRANE_initialize`.

`inform` is a scalar `INTENT(OUT)` argument of type `FILTRANE_inform_type` (see Section 2.4.3). 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 2.6 and 2.7.

`data` is a scalar `INTENT(OUT)` argument of type `FILTRANE_data_type` (see Section 2.4.4). 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 2.6).

2.5.2 The subroutine that applies the `FILTRANE` algorithm to the problem

The `FILTRANE` solver is called as follows:

```
        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`, `x_l`, `x_u`, `x_status`, `c_l`, `c_u`, `J_size`, `J_type` and `infinity` components must be set. In addition, its `c`, `y`, `g`, `equation`, `J_val`, `J_row` and `J_col` components must be allocated (see Section 2.4.1).

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 θ at the point `x`,
`g`	contains the gradient of θ at the point `x`,

If `problem%m` > 0, then

`c`	contains the values of the constraints at the point `x`,

If, additionally, `control%external_Jacobian_products` is `.FALSE.`, then

`J_val` contains the values of the nonzero entries of the constraints Jacobian at `x`,

`J_row` contains the row indices of the the nonzero entries of the constraints Jacobian at `x`,

`J_col` contains the column indices of the the nonzero entries of the constraints Jacobian at `x`.

`control` is a scalar `INTENT(IN)` argument of type `FILTRANE_control_type` (see Section 2.4.2). Default values may be assigned by calling `FILTRANE_initialize` prior to the first call to `FILTRANE_solve`.

`info` is a scalar `INTENT(OUT)` argument of type `FILTRANE_inform_type` (see Section 2.4.3). A successful call to the routine `FILTRANE_solve` is indicated when the component `status` has the value 0. For other return values of `status`, see Section 2.7.

`data` is a scalar `INTENT(INOUT)` argument of type `FILTRANE_data_type` (see Section 2.4.4). 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 `FILTRANE` routines, except for returning values to FILTRANE_solve via the reverse communication interface (see Section 2.6).

2.5.3 The termination subroutine

All previously allocated workspace arrays for FILTRANE are deallocated as follows:

```
CALL FILTRANE_terminate( control, info, data )
```

`control` is a scalar `INTENT(IN)` argument of type `FILTRANE_control_type` exactly as for `FILTRANE_initialize`.

`info` is a scalar `INTENT(OUT)` argument of type `FILTRANE_inform_type` exactly as for `FILTRANE_initialize`. A successful call to `FILTRANE_terminate` is indicated when the component `status` has the value 0. For other return values of `status`, see Section 2.7.

`data` is a scalar `INTENT(INOUT)` argument of type `FILTRANE_data_type` exactly as for `FILTRANE_solve`, which must not have been altered by the user since the last call to `FILTRANE_initialize`, except for returning values to FILTRANE_solve via the reverse communication interface (see Section 2.6). On exit, array components will have been deallocated.

Note that a call to this routine is mandatory before `FILTRANE_solve` is called for a new problem whose structure differs from the current one.

2.6 Reverse communication

A positive value of `info%status` on exit from `FILTRANE_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 `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)`,
- 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 $\mathbf{J}(x)\mathbf{v}$, where $\mathbf{v}$ 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 2.4.2).

`8, 9, 10 and 11.` The user must compute the product $\mathbf{J}(x)^T\mathbf{v}$, where $\mathbf{v}$ is given by `data%RC_v(1:problem%m)` and place the result in the vector `data%RC_Mv(1:problem%n)`. 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 2.4.2).

`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

$$\frac{\alpha(\alpha-1)}{2}\left\{\mathbf{J}(\mathbf{x})\mathrm{diag}\left([\theta_c(\mathbf{x})]_i^{\alpha-2}\right)\mathbf{J}(\mathbf{x})^T+\mathrm{diag}\left([\theta_x(\mathbf{x})]_i^{\alpha-2}\right)\right\}+\frac{\alpha}{2}\sum_{i=1}^{m}[\theta_c(\mathbf{x})]_i^{\alpha-1}\nabla^2_{xx}\mathbf{c}_i(\mathbf{x}),$$

where $\theta_c(x)$ contains the first m components of $\theta(x)$ (the constraints's violations) and $\theta_x(x)$ the last n (the bounds' violations), where α 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 2.4.2), 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 2.4.2).

`15 and 16.` The user must compute the product

$$\sum_{i=1}^{m}\mathbf{y}_i\nabla_{xx}\mathbf{c}_i(\mathbf{x})\mathbf{v},$$

where

- $\mathbf{y}$ is given by `problem%y(1:problem%m)`,

- **x** is given by `problem%x(1:problem%n)`,
- **v** is given by `data%RC_v(1:problem%n)`,

and place the result in the vector `data%RC_Mv(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%RC_newx` 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 2.4.2).

2.7 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.

`-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.)

2.8 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 2.4.2), 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:

```
( .. lines ignored by FILTRANE_read_specfile .. )
  BEGIN FILTRANE
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by FILTRANE_read_specfile .. )
```

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 parameters 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", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE". 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 2.3).

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.

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

`control` is a scalar `INTENT(INOUT)` argument of type `FILTRANE_control_type` (see Section 2.4.2). 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 2.4.2) 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 2.4.3).

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

Table 2.1: Specfile commands and associated components of `control` (part 1).

2.9 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 ρ 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:

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

Table 2.2: Specfile commands and associated components of `control` (part 2).

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.

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 $\theta(\mathbf{x})$ 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:

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\text{solve}} \rightarrow \boxed{\text{terminate}} \right]$$

or

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\text{read specfile}} \rightarrow \left[\boxed{\text{solve}} \rightarrow \boxed{\text{terminate}} \right] \right]$$

where the procedure's control parameter may be modified by reading the specfile (see Section 2.8). Each of the "boxed" steps in these sequences corresponds to calling a specific routine of the package (see Section 2.5). 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, S. Leyffer and Ph. L. Toint (2003), A Multidimensional Filter Algorithm for Nonlinear Equations and Nonlinear Least-Squares, Technical report 03/01, The University of Namur, Belgium.

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.

5 EXAMPLE OF USE

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

$$3\mathbf{x}_1^2 + 2\mathbf{x}_2^3 + \mathbf{x}_1\mathbf{x}_2 = 0,$$

$$\mathbf{x}_1 + \mathbf{x}_2 = 0,$$

$$-2 \leq \mathbf{x}_1 \leq 2 \text{ and } -2 \leq \mathbf{x}_2 \leq 2,$$

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

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} 6\mathbf{x}_1 + \mathbf{x}_2 & \mathbf{x}_1 + 6\mathbf{x}_2^2 \\ 1 & 1 \end{pmatrix}.$$

We may then use the following code:

```
! 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 )
! 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_val( 3 ) = 6.0D0 * problem%x( 2 ) ** 2 + problem%x( 1 )
        problem%J_val( 4 ) = 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
!       Note that H2, the Hessian of C2 is identically zero, since this
!       constraint is linear. Hence the terms in y(2)*H2 disappear.
        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( 1 ) * H1( 2 ) * FILTRANE_data%RC_v( 2 )
        FILTRANE_data%RC_Mv( 2 ) =                                      &
                        problem%y( 1 ) * H1( 2 ) * FILTRANE_data%RC_v( 1 ) +   &
                        problem%y( 1 ) * H1( 3 ) * 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.
  WRITE( iout, 1000 )
  WRITE( iout, 1001 )
  WRITE( iout, 1000 )
  WRITE( iout, 1002 ) problem%x( 1 )
  WRITE( iout, 1003 ) problem%x( 2 )
  WRITE( iout, 1000 )
  WRITE( iout, 1004 ) problem%c( 1 )
  WRITE( iout, 1005 ) problem%c( 2 )
  WRITE( iout, 1000 )
  WRITE( iout, 1006 ) FILTRANE_inform%status
  WRITE( iout, 1007 ) problem%f
  WRITE( iout, 1008 ) FILTRANE_inform%nbr_iterations,                   &
                      FILTRANE_inform%nbr_cg_iterations
  WRITE( iout, 1009 ) FILTRANE_inform%nbr_c_evaluations
  WRITE( iout, 1010 ) FILTRANE_inform%nbr_J_evaluations
! Cleanup the problem.
  CALL NLPT_cleanup( problem )
  STOP
```

```
! Formats
1000 FORMAT(/)
1001 FORMAT(' Problem : GALEXAMPLE')
1002 FORMAT(' X1 = ',1PE20.12)
1003 FORMAT(' X2 = ',1PE20.12)
1004 FORMAT(' C1 = ',1PE20.12)
1005 FORMAT(' C2 = ',1PE20.12)
1006 FORMAT(' Exit condition number    = ',i10)
1007 FORMAT(' Objective function value =',1PE20.12)
1008 FORMAT(' Number of iterations     = ',i10,/,                             &
            ' Number of CG iterations  = ',i10)
1009 FORMAT(' Number of constraints evaluations =',i6)
1010 FORMAT(' Number of Jacobian evaluations    =',i6)
END PROGRAM GALAHAD_FILTRANE_EXAMPLE
```

This produces the following output:

```
              **************************************************
              *                                                *
              *                    FILTRANE                    *
              *                                                *
              *     GALAHAD filter trust-region algorithm      *
              *                                                *
              *     for the nonlinear feasibility problem      *
              *                                                *
              **************************************************


 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+00  8.965E+00  9.341E-01  6.223E-01  1.000E+00      1 GUIF   0
    2   6.430E-01  1.538E+01  6.644E-01  2.033E+00  1.245E+00      3 GUEf   1
    3   7.325E-03  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-05  3.180E-03  9.999E-01  8.230E-03  1.245E+00      7 GUIF   1
    6   8.208E-07  1.103E-02  9.878E-01  4.878E-02  1.245E+00      9 GUIF   1
    7   2.099E-08  4.705E-05  1.000E+00  1.450E-04  1.245E+00     10 GUIF   1
    8   9.619E-14  3.773E-06  1.000E+00  8.922E-04  1.245E+00     12 GUIF   1

 Problem successfully solved: constraints violations are small.


 Problem : GALEXAMPLE


 X1 =   1.000000219305E+00
 X2 =  -1.000000219305E+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

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

are replaced by

```
! open specfile
   OPEN( 57, FILE = FILTRANE.SPC', STATUS = 'OLD' )
! read its content (asking for some output)
   CALL FILTRANE_read_specfile( 57, control, inform )
! close it
   CLOSE( 57 )
```

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
```