GALAHAD WCP

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

1 SUMMARY

This package uses a primal-dual interior-point method to **find a well-centered interior point x** for a set of general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i = 1, \ldots, m, \tag{1.1}$$

and simple bounds

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n, \tag{1.2}$$

where the vectors $\mathbf{a}_i$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$ and $\mathbf{x}^u$ are given. More specifically, if possible, the package finds a solution to the system of primal optimality equations

$$\mathbf{A}\mathbf{x} = \mathbf{c}, \tag{1.3}$$

dual optimality equations

$$\mathbf{g} = \mathbf{A}^T \mathbf{y} + \mathbf{z}, \ \ \mathbf{y} = \mathbf{y}^l + \mathbf{y}^u \ \text{ and } \ \mathbf{z} = \mathbf{z}^l + \mathbf{z}^u, \tag{1.4}$$

and perturbed complementary slackness equations

$$(c_i - c_i^l) y_i^l = (\mu_c^l)_i \ \text{ and } \ (c_i - c_i^u) y_i^u = (\mu_c^u)_i, \quad i = 1, \ldots, m, \tag{1.5}$$

and

$$((x_j - x_j^l) z_j^l = (\mu_x^l)_j \ \text{ and } \ (x_j - x_j^u) z_j^u = (\mu_x^u)_j, \quad j = 1, \ldots, n, \tag{1.6}$$

for which

$$\mathbf{c}^l \leq \mathbf{c} \leq \mathbf{c}^u, \ \ \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u, \ \ \mathbf{y}^l \geq \mathbf{0}, \ \ \mathbf{y}^u \leq \mathbf{0}, \ \ \mathbf{z}^l \geq \mathbf{0} \ \text{ and } \ \mathbf{z}^u \leq \mathbf{0}. \tag{1.7}$$

Here $\mathbf{A}$ is the matrix whose rows are the $\mathbf{a}_i^T$, $i = 1, \ldots, m$, $\boldsymbol{\mu}_c^l$, $\boldsymbol{\mu}_c^u$, $\boldsymbol{\mu}_x^l$ and $\boldsymbol{\mu}_x^u$ are vectors of strictly positive *targets*, $\mathbf{g}$ is another given target vector (which is often zero), and $(\mathbf{y}^l, \mathbf{y}^u)$ and $(\mathbf{z}^l, \mathbf{z}^u)$ are dual variables for the linear constraints and simple bounds respectively; $\mathbf{c}$ gives the constraint value $\mathbf{A}\mathbf{x}$. Since (1.5)–(1.7) normally imply that

$$\mathbf{c}^l < \mathbf{c} < \mathbf{c}^u, \ \ \mathbf{x}^l < \mathbf{x} < \mathbf{x}^u, \ \ \mathbf{y}^l > \mathbf{0}, \ \ \mathbf{y}^u < \mathbf{0}, \ \ \mathbf{z}^l > \mathbf{0} \ \text{ and } \ \mathbf{z}^u < \mathbf{0}, \tag{1.8}$$

such a primal-dual point $(\mathbf{x}, \mathbf{c}, \mathbf{y}^l, \mathbf{y}^u, \mathbf{z}^l, \mathbf{z}^l)$ may be used, for example, as a feasible starting point for primal-dual interior-point methods for solving the linear programming problem of minimizing $\mathbf{g}^T \mathbf{x}$ subject to (1.1) and (1.2).

Full advantage is taken of any zero coefficients in the vectors $\mathbf{a}_i$. Any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite. The package identifies infeasible problems, and problems for which there is no strict interior, that is one or more of (1.7) only holds as an equality for all feasible points.

ATTRIBUTES — Versions: `GALAHAD_WCP_single, GALAHAD_WCP_double`. **Uses:** `GALAHAD_CLOCK`, `GALAHAD_SYMBOLS`, `GALAHAD_SMT`, `GALAHAD_QPT`, `GALAHAD_QPD`, `GALAHAD_SPECFILE`, `GALAHAD_QPP`, `GALAHAD_SPACE`, `GALAHAD_SORT`, `GALAHAD_STRING`, `GALAHAD_ROOTS`, `GALAHAD_SBLS`, `GALAHAD_FDC`. **Date:** July 2006. **Origin:** C. Cartis and N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Parallelism:** Some options may use OpenMP and its runtime library.

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_WCP_single
```

with the obvious substitution `GALAHAD_WCP_double`, `GALAHAD_WCP_single_64` and `GALAHAD_WCP_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 `SMT_type`, `QPT_problem_type`, `WCP_time_type`, `WCP_control_type`, `WCP_inform_type` and `WCP_data_type` (Section 2.4) and the subroutines `WCP_initialize`, `WCP_solve`, `WCP_terminate`, (Section 2.5) and `WCP_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

The constraint Jacobian $\mathbf{A}$, that is, the matrix whose rows are the vectors $\mathbf{a}_i^T$, $i = 1,\ldots,m$, may be stored in one of three input formats.

2.1.1 Dense storage format

The matrix is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Component $n*(i-1)+j$ of the storage array `A%val` will hold the value a_{ij} for $i = 1,\ldots,m$, $j = 1,\ldots,n$.

2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrix are stored. For the l-th entry of $\mathbf{A}$, its row index i, column index j and value a_{ij} are stored in the l-th components of the integer arrays `A%row`, `A%col` and real array `A%val`, respectively. The order is unimportant, but the total number of entries `A%ne` is also required.

2.1.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row $i+1$. For the i-th row of $\mathbf{A}$, the i-th component of a integer array `A%ptr` holds the position of the first entry in this row, while `A%ptr` $(m+1)$ holds the total number of entries plus one. The column indices j and values a_{ij} of the entries in the i-th row are stored in components $l =$ `A%ptr`(i), ...,`A%ptr` $(i+1)-1$ of the integer array `A%col`, and real array `A%val`, respectively.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

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 Parallel usage

OpenMP may be used by the GALAHAD_WCP package to provide parallelism for some solvers in shared memory environments. See the documentation for the GALAHAD package SLS for more details. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp). The number of threads may be controlled at runtime by setting the environment variable OMP_NUM_THREADS.

MPI may also be used by the package to provide parallelism for some solvers in a distributed memory environment. To use this form of parallelism, MPI must be enabled at runtime by using the correct compiler flag (usually some variant of -lmpi). Although the MPI process will be started automatically when required, it should be stopped by the calling program once no further use of this form of parallelism is needed. Typically, this will be via statements of the form

```
CALL MPI_INITIALIZED( flag, ierr )
IF ( flag ) CALL MPI_FINALIZE( ierr )
```

The code may be compiled and run in serial mode.

2.4 The derived data types

Six derived data types are accessible from the package.

2.4.1 The derived data type for holding matrices

The derived data type SMT_TYPE is used to hold the matrix $\mathbf{A}$. The components of SMT_TYPE used here are:

m is a scalar component of type INTEGER(ip_), that holds the number of rows in the matrix.

n is a scalar component of type INTEGER(ip_), that holds the number of columns in the matrix.

ne is a scalar variable of type INTEGER(ip_), that holds the number of matrix entries

type is a rank-one allocatable array of type default CHARACTER, that is used to indicate the matrix storage scheme used. Its precise length and content depends on the type of matrix to be stored (see §2.4.2).

val is a rank-one allocatable array of type REAL(rp_) and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $a_{ij} = a_{ji}$ of a matrix $\mathbf{A}$ is represented as a single entry (see §2.1.1–2.1.3). Any duplicated entries that appear in the sparse co-ordinate or row-wise schemes will be summed.

row is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the row indices of the entries. (see §2.1.2).

col is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the column indices of the entries (see §2.1.2–2.1.3).

ptr is a rank-one allocatable array of type INTEGER(ip_), and dimension at least m + 1, that may hold the pointers to the first entry in each row (see §2.1.3).

2.4.2 The derived data type for holding the problem

The derived data type QPT_problem_type is used to hold the problem. The components of QPT_problem_type are:

new_problem_structure is a scalar variable of type default LOGICAL, that is .TRUE. if this is the first (or only) problem in a sequence of problems with identical "structure" to be attempted, and .FALSE. if a previous problem with the same "structure" (but different numerical data) has been solved. Here, the term "structure" refers both to the sparsity patterns of the Jacobian matrices $\mathbf{A}$ involved (but not their numerical values), to the zero/nonzero/infinity patterns (a bound is either zero, $\pm$ infinity, or a finite but arbitrary nonzero) of each of the constraint bounds, and to the variables and constraints that are fixed (both bounds are the same) or free (the lower and upper bounds are $\pm$ infinity, respectively).

n is a scalar variable of type INTEGER(ip_), that holds the number of optimization variables, n.

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

gradient_kind is a scalar variable of type INTEGER(ip_), that is used to indicate whether the components of the target vector $\mathbf{g}$ have special or general values. Possible values for gradient_kind are:

0 In this case, $\mathbf{g} = 0$.

1 In this case, $g_i = 1$ for $i = 1, \ldots, n$.

$\neq$ 0,1 In this case, general values of $\mathbf{g}$ will be used, and will be provided by the user in the component G.

G is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the target vector $\mathbf{g}$. The j-th component of G, $j = 1, \ldots, n$, contains $\mathbf{g}_j$. If gradient_kind = 0, 1, G need not be allocated.

A is scalar variable of type SMT_TYPE that holds the Jacobian matrix $\mathbf{A}$. The following components are used:

A%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of A%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of A%type must contain the string COORDINATE, while for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of A%type must contain the string SPARSE_BY_ROWS.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into A%type. For example, if prob is of derived type WCP_problem_type and involves a Jacobian we wish to store using the sparse row-wise storage scheme, we may simply

```
CALL SMT_put( prob%A%type, 'SPARSE_BY_ROWS' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

A%ne is a scalar variable of type INTEGER(ip_), that holds the number of entries in $\mathbf{A}$ in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for either of the other two schemes.

A%val is a rank-one allocatable array of type REAL(rp_), that holds the values of the entries of the Jacobian matrix $\mathbf{A}$ in any of the storage schemes discussed in Section 2.1.

A%row is a rank-one allocatable array of type INTEGER(ip_), that holds the row indices of $\mathbf{A}$ in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be allocated for either of the other two schemes.

A%col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of $\mathbf{A}$ in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated when the dense storage scheme is used.

A%ptr is a rank-one allocatable array of dimension m+1 and type INTEGER(ip_), that holds the starting position of each row of $\mathbf{A}$, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3). It need not be allocated when the other schemes are used.

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, where infinity is a component of the control array control (see Section 2.4.3).

C_u is a rank-one allocatable array of dimension m and type REAL(rp_), that holds the vector of upper bounds $\mathbf{c}^u$ 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, where infinity is a component of the control array control (see Section 2.4.3).

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 the 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 -infinity, where infinity is a component of the control array control (see Section 2.4.3).

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 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, where infinity is a component of the control array control (see Section 2.4.3).

X is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the values $\mathbf{x}$ of the optimization variables. The j-th component of X, $j = 1,\ldots,n$, contains x_j.

Z_l is a rank-one allocatable array of dimension n and type default REAL(rp_), that holds the values $\mathbf{z}^l$ of estimates of the dual variables corresponding to the lower simple bound constraints $\mathbf{x}^l \leq \mathbf{x}$ (see equation (1.4)). The j-th component of Z_l, $j = 1,\ldots,n$, contains z^l_j.

Z_u is a rank-one allocatable array of dimension n and type default REAL(rp_), that holds the values $\mathbf{z}^u$ of estimates of the dual variables corresponding to the upper simple bound constraints $\mathbf{x} \leq \mathbf{x}^u$ (see equation (1.4)). The j-th component of Z_l, $j = 1,\ldots,n$, contains z^l_j.

C is a rank-one allocatable array of dimension m and type default REAL(rp_), that holds the values $\mathbf{Ax}$ of the constraints. The i-th component of C, $i = 1,\ldots,m$, contains $\mathbf{a}_i^T\mathbf{x} \equiv (\mathbf{Ax})_i$.

Y_l is a rank-one allocatable array of dimension m and type default REAL(rp_), that holds the values $\mathbf{y}^l$ of estimates of the Lagrange multipliers corresponding to the lower general constraints $\mathbf{c}^l \leq \mathbf{Ax}$ (see equation (1.4)). The i-th component of Y_l, $i = 1,\ldots,m$, contains y^l_i.

Y_u is a rank-one allocatable array of dimension m and type default REAL(rp_), that holds the values $\mathbf{y}^u$ of estimates of the Lagrange multipliers corresponding to the upper general constraints $\mathbf{Ax} \leq \mathbf{c}^u$ (see equation (1.4)). The i-th component of Y_u, $i = 1,\ldots,m$, contains y^u_i.

2.4.3 The derived data type for holding control parameters

The derived data type WCP_control_type is used to hold controlling data. Default values may be obtained by calling WCP_initialize (see Section 2.5.1), while components may also be changed by calling GALAHAD_WCP_read_spec (see Section 2.7.1). The components of WCP_control_type are:

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

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

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

maxit is a scalar variable of type INTEGER(ip_), that holds the maximum number of iterations which will be allowed in WCP_solve. The default is maxit = 1000.

start_print is a scalar variable of type INTEGER(ip_), that specifies the first iteration for which printing will occur in WCP_solve. If start_print is negative, printing will occur from the outset. The default is start_print = -1.

stop_print is a scalar variable of type INTEGER(ip_), that specifies the last iteration for which printing will occur in WCP_solve. If stop_print is negative, printing will occur once it has been started by start_print. The default is stop_print = -1.

initial_point is a scalar variable of type INTEGER(ip_), that indicates how the initial point is chosen. Possible values are:

- 0 the input point $\mathbf{x}$ may be perturbed to ensure that its j-th component is feasible with respect to its bounds $x_j^l \leq x_j \leq x_j^u$, and if possible at least prfeas from either bound. If possible, the slack variable $\mathbf{c}$ will be feasible and at least prfeas from its bounds $\mathbf{c}^l \leq \mathbf{c} \leq \mathbf{c}^u$. The input dual variables/Lagrange multipliers will be perturbed so that they are feasible and at least dufeas from their bounds $\mathbf{y}^l \geq 0$, $\mathbf{y}^u \leq 0$, $\mathbf{z}^l \geq 0$ and $\mathbf{z}^u \leq 0$. The remaining constraints (1.3)–(1.6) will most likely not be satisfied. The feasible region is that enlarged by perturbing all inequality constraints by perturb_start (see below and §4).
- 1 a point satisfying (1.3)–(1.4) that also tries to satisfy (1.8) will be computed. The feasible region is enlarged by perturbing each inequality constraints so that its residual is at least prfeas (primal and slack variables) or dufeas (dual variables and Lagrange multipliers).
- 2 as in 1, but the perturbations are equal and chosen so that the smallest residual is at least prfeas (primal and slack variables) or dufeas (dual variables and Lagrange multipliers).

The default is initial_point = 0.

perturbation_strategy is a scalar variable of type INTEGER(ip_), that specifies the strategy used for perturbing and then tightening the inequality constraints. The perturbations are chosen so that the iterates lie interior to the perturbed constraints (see §4). Possible values are:

- $\leq$ 0 no perturbation is allowed.
- 1 the initial perturbation is determined by the initial-point strategy (see initial_point above). The perturbation θ, say, on a generic constraint $w \geq w^l$ is subsequently reduced to $\bar{\theta}$ so that $w - w^l + \bar{\theta} = (1-\eta)(w - w^l + \theta)$ where η has the value frac_perturb.
- 2 the initial perturbation is determined by the initial-point strategy (see initial_point above). The perturbation θ, say, on a generic constraint $w \geq w^l$ is subsequently reduced to $\bar{\theta}$ so that $w - w^l + \bar{\theta} = (1-\eta)(w - w^l + \theta)$, where η has the value frac_perturb, whenever $w \leq w^l$. If $w > w^l$, the perturbation $\bar{\theta}$ is set to zero.
- 3 the same as 1, except that $\eta \leq$ frac_perturb is reduced gradually to zero.
- 4 the same as 2, except that $\eta \leq$ frac_perturb is reduced gradually to zero.

The default is perturbation_strategy = 2.

infeas_max is a scalar variable of type INTEGER(ip_), that specifies the number of iterations for which the overall infeasibility of the problem is not reduced by at least a factor required_infeas_reduction before the problem is flagged as infeasible (see required_infeas_reduction). The default is infeas_max = 200.

restore_problem is a scalar variable of type INTEGER(ip_), that specifies how much of the input problem is to be retored on output. Possible values are:

0 nothing is restored.

1 the vector data $\mathbf{w}$, $\mathbf{g}$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ will be restored to their input values.

2 the entire problem, that is the above vector data along with the Jacobian matrix $\mathbf{A}$, will be restored.

The default is `restore_problem = 2`.

`infinity` is a scalar variable of type `REAL(rp_)`, that is used to specify which constraint bounds are infinite. Any bound larger than `infinity` in modulus will be regarded as infinite. The default is `infinity` = 10^{19}.

`stop_p` is a scalar variable of type `REAL(rp_)`, that holds the required accuracy for the primal infeasibility (1.3). The default is `stop_p` = $u^{1/3}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_WCP_double`).

`stop_d` is a scalar variable of type default `REAL(rp_)`, that holds the required accuracy for the dual infeasibility (1.4) The default is `stop_d` = $u^{1/3}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_WCP_double`).

`stop_c` is a scalar variable of type default `REAL(rp_)`, that holds the required accuracy for the violation of perturbed complementarity slackness (1.5)–(1.6). The default is `stop_c` = $u^{1/3}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_WCP_double`).

`perturb_start` is a scalar variable of type `REAL(rp_)`, that is used to assign the initial value of the perturbations to each bound constraint ((see §4) whenever the initial point strategy `initial_point = 0` is used (see above). If `perturb_start`≤ 0, a suitable value will be computed by the package. The default is `perturb_start = - 1.0`.

`reduce_perturb_factor` is a scalar variable of type `REAL(rp_)`, that is used to adjust the perturbations. This corresponds to the value ξ in §4, and should lie strictly between zero and one. Any value outside $(0,1)$ will be reset to a suitable value by the package. The default is `reduce_perturb_factor = 0.25`.

`reduce_perturb_multiplier` is a scalar variable of type `REAL(rp_)`, that is used to adjust the perturbations. This corresponds to the value ρ in §4, and should lie strictly between zero and one. Any value outside $(0,1)$ will be reset to a suitable value by The default is `reduce_perturb_factor = 0.01`.

`insufficiently_feasible` is a scalar variable of type `REAL(rp_)`, that is used to adjust the perturbations. This corresponds to the value ε in §4, and should be strictly positive. Any value non-negative value will be reset to a suitable value by the package. The default is `insufficiently_feasible` = $u^{1/4}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_WCP_double`).

`implicit_tol` is a scalar variable of type `REAL(rp_)`, that is used to assess whether a constraint defines an implicit equality (see §4). Any of the constraints (1.7) that is less feasible than `implicit_tol` will be regarded as defining an implicit equality. The default is `implicit_tol` = $u^{1/3}$, where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_WCP_double`).

`perturbation_small` is a scalar variable of type `REAL(rp_)`, that is used to determine whether the problem is feasible, but not an interior, point. If the maximum constraint perturbation is strictly positive but smaller than `perturbation_small` and the maximum violation of (1.7) is smaller than `implicit_tol`, the method will deduce that there is a feasible point but no interior. If `perturbation_small`≤ 0, a suitable value will be computed by the package. The default is `perturbation_small = - 1.0`.

`prfeas` is a scalar variable of type `REAL(rp_)`, that aims to specify the closest that any initial variable may be to infeasibility. Any variable closer to infeasibility than `prfeas` will be moved to `prfeas` from the offending bound. However, if a variable is range bounded, and its bounds are closer than `prfeas` apart, it will be moved to the mid-point of the two bounds. The default is `prfeas = 1.0`.

dufeas is a scalar variable of type REAL(rp_), that aims to specify the closest that any initial dual variable or Lagrange multiplier may be to infeasibility. Any variable closer to infeasibility than prfeas will be moved to dufeas from the offending bound. However, if a dual variable is range bounded, and its bounds are closer than dufeas apart, it will be moved to the mid-point of the two bounds. The default is dufeas = 1.0.

required_infeas_reduction is a scalar variable of type default REAL(rp_), that specifies the least factor by which the overall infeasibility of the problem must be reduced, over infeas_max consecutive iterations, for it not be declared infeasible (see infeas_max). The default is required_infeas_reduction = 0.99.

alpha_scale is a scalar variable of type REAL(rp_), that is used to specify the scaling factor $\alpha > 0$ that is used to assess dependent constraints (see §4). Any non-positive value will be reset by the package to the default. The default is alpha_scale = 0.01.

identical_bounds_tol is a scalar variable of type REAL(rp_). Every pair of constraint bounds (c_i^l, c_i^u) or (x_j^l, x_j^u) that is closer than identical_bounds_tol will be reset to the average of their values, $\frac{1}{2}(c_i^l + c_i^u)$ or $\frac{1}{2}(x_j^l + x_j^u)$ respectively. The default is identical_bounds_tol = u, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_WCP_double).

mu_target is a scalar variable of type REAL(rp_), that gives the initial value for each of the targets $\boldsymbol{\mu}_c^l$, $\boldsymbol{\mu}_c^u$, $\boldsymbol{\mu}_x^l$ and $\boldsymbol{\mu}_x^u$. If mu_target_start$\leq$ 0, a suitable value will be computed by the package. The default is mu_target = - 1.0.

mu_accept_fraction is a scalar variable of type REAL(rp_), is used to allow flexibility when attempting to satisfy the perturbed complementarity equations (1.5)–(1.6). The value corresponds to the parameter $\gamma > 0$ in the termination condition (4.4) (see §4). If mu_accept_fraction$\leq$ 0, a suitable positive value will be computed by the package. The default is mu_accept_fraction = 1.0.

mu_increase_factor is a scalar variable of type REAL(rp_), that is used to specify the value $\beta \geq 1$ by which the targets corresponding to nonzero bound perturbations will be increased (see §4). Any value smaller than one will be reset to one. The default is mu_increase_factor = 2.0.

cpu_time_limit is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted CPU time. Any negative value indicates no limit will be imposed. The default is cpu_time_limit = - 1.0.

clock_time_limit is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted elapsed system clock time. Any negative value indicates no limit will be imposed. The default is clock_time_limit = - 1.0.

remove_dependencies is a scalar variable of type default LOGICAL, that must be set .TRUE. if the algorithm is to attempt to remove any linearly dependent constraints before solving the problem, and .FALSE. otherwise. We recommend removing linearly dependencies. The default is remove_dependencies = .TRUE..

treat_zero_bounds_as_general is a scalar variable of type default LOGICAL. If it is set to .FALSE., variables which are only bounded on one side, and whose bound is zero, will be recognised as non-negativities/non-positivities rather than simply as lower- or upper-bounded variables. If it is set to .TRUE., any variable bound x_j^l or x_j^u which has the value 0.0 will be treated as if it had a general value. Setting treat_zero_bounds_as_general to .TRUE. has the advantage that if a sequence of problems are reordered, then bounds which are "accidentally" zero will be considered to have the same structure as those which are nonzero. However, GALAHAD_WCP is able to take special advantage of non-negativities/non-positivities, so if a single problem, or if a sequence of problems whose bound structure is known not to change, is/are to be solved, it will pay to set the variable to .FALSE.. The default is treat_zero_bounds_as_general = .FALSE..

just_feasible is a scalar variable of type default LOGICAL, that must be set .TRUE. if the algorithm should stop as soon as a feasible interior point of the constraint set is found. Otherwise a a well-centered interior point will be sought The default is just_feasible = .FALSE..

balance_initial_complementarity is a scalar variable of type default LOGICAL, that must be set .TRUE. if the initial dual variables and Lagrange multipliers are to be reset to ensure that the perturbed complementarity (4.1)–(4.3) is satisfied (see §4), and .FALSE. otherwise. The default is balance_initial_complementarity = .FALSE..

use_corrector is a scalar variable of type default LOGICAL, that must be set .TRUE. if a corrector step $\ddot{\mathbf{v}}$ is to be used (see §4), and .FALSE. otherwise. The default is use_corrector = .FALSE..

record_x_status is a scalar variable of type default LOGICAL, that must be set .TRUE. if the array inform%X_status (see §2.4.5) is to be allocated and the status of the bound constraints reported on exit. Otherwise, inform%X_status will not be allocated. The default is record_x_status = .TRUE..

record_c_status is a scalar variable of type default LOGICAL, that must be set .TRUE. if the array inform%C_status (see §2.4.5) is to be allocated and the status of the bound constraints reported on exit. Otherwise, inform%C_status will not be allocated. The default is record_c_status = .TRUE..

space_critical is a scalar variable of type default LOGICAL, that must be set .TRUE. if space is critical when allocating arrays and .FALSE. otherwise. The package may run faster if space_critical is .FALSE. but at the possible expense of a larger storage requirement. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

feasol is a scalar variable of type default LOGICAL, that should be set .TRUE. if the final solution obtained will be perturbed so that variables close to their bounds are moved onto these bounds, and .FALSE. otherwise. The default is feasol = .FALSE..

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

FDC_control is a scalar variable of type FDC_control_type whose components are used to control any detection of linear dependencies performed by the package GALAHAD_FDC. See the specification sheet for the package GALAHAD_FDC for details, and appropriate default values.

SBLS_control is a scalar variable of type SBLS_control_type whose components are used to control factorizations performed by the package GALAHAD_SBLS. See the specification sheet for the package GALAHAD_SBLS for details, and appropriate default values.

2.4.4 The derived data type for holding timing information

The derived data type WCP_time_type is used to hold elapsed CPU and system clock times for the various parts of the calculation. The components of WCP_time_type are:

total is a scalar variable of type REAL(rp_), that gives the total CPU time spent in the package.

preprocess is a scalar variable of type REAL(rp_), that gives the CPU time spent reordering the problem to standard form prior to solution.

find_dependent is a scalar variable of type REAL(rp_), that gives the CPU time spent detecting and removing linearly-dependent equality constraints

analyse is a scalar variable of type REAL(rp_), that gives the CPU time spent analysing the required matrices prior to factorization.

factorize is a scalar variable of type REAL(rp_), that gives the CPU time spent factorizing the required matrices.

solve is a scalar variable of type REAL(rp_), that gives the CPU time spent computing the search direction.

clock_total is a scalar variable of type REAL(rp_), that gives the total elapsed system clock time spent in the package.

clock_preprocess is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent reordering the problem to standard form prior to solution.

clock_find_dependent is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent detecting and removing linearly-dependent equality constraints

clock_analyse is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent analysing the required matrices prior to factorization.

clock_factorize is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent factorizing the required matrices.

clock_solve is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent computing the search direction.

2.4.5 The derived data type for holding informational parameters

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

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Section 2.6 for details.

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

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

iter is a scalar variable of type INTEGER(ip_), that gives the total number of iterations required.

factorization_status is a scalar variable of type INTEGER(ip_), that gives the return status from the matrix factorization.

factorization_integer is a scalar variable of type long INTEGER(ip_), that gives the amount of integer storage used for the matrix factorization.

factorization_real is a scalar variable of type INTEGER(int64), that gives the amount of real storage used for the matrix factorization.

c_implicit is a scalar variable of type INTEGER(ip_), that gives the number of variables **c** that lie on (one) of their bounds for all feasible solutions to (1.3)–(1.7). Each of these variables implies that the corresponding value of the constraint **Ax** may be fixed at the appropriate bound, and the constraint subsequently treated as an equality. See C_status.

x_implicit is a scalar variable of type INTEGER(ip_), that gives the number of variables $\mathbf{x}$ that lie on (one) of their bounds for all feasible solutions to (1.3)–(1.7). Each of these variables may then be fixed at its appropriate bound, and the variable subsequently removed from the problem. See X_status.

y_implicit is a scalar variable of type INTEGER(ip_), that gives the number of Lagrange multipliers $\mathbf{y}^l$ and $\mathbf{y}^u$ that lie on their bounds for all feasible solutions to (1.3)–(1.7). Each of the corresponding constraints $\mathbf{A}\mathbf{x} \geq \mathbf{c}^l$ and/or $\mathbf{A}\mathbf{x} \leq \mathbf{c}^u$ may subsequently be removed from the problem. See C_status.

z_implicit is a scalar variable of type INTEGER(ip_), that gives the number of dual variables $\mathbf{z}^l$ and $\mathbf{z}^u$ that lie on their bounds for all feasible solutions to (1.3)–(1.7). Each of the corresponding simple bounds $\mathbf{x} \geq \mathbf{x}^l$ and/or $\mathbf{x} \leq \mathbf{x}^u$ may subsequently be removed from the problem. See X_status.

obj is a scalar variable of type REAL(rp_), that holds the value of the (primal) objective function, $\mathbf{g}^T\mathbf{x}$, at the well-centered point found.

mu_final_target_max is a scalar variable of type REAL(rp_), that holds the largest component of all the targets $\boldsymbol{\mu}_c^l$, $\boldsymbol{\mu}_c^u$, $\boldsymbol{\mu}_x^l$ and $\boldsymbol{\mu}_x^u$ used at the well-centered point found.

non_negligible_pivot is a scalar variable of type REAL(rp_), that holds the value of the smallest pivot larger than control%zero_pivot when searching for dependent linear constraints. If non_negligible_pivot is close to control%zero_pivot, this may indicate that there are further dependent constraints, and it may be worth increasing control%zero_pivot above non_negligible_pivot and solving again.

feasible is a scalar variable of type default LOGICAL, that has the value .TRUE. if the output values of $\mathbf{x}$ and the Lagrange multipliers and dual variables lie in the strict interior of the primal-dual feasible region, and the value .FALSE. otherwise.

time is a scalar variable of type WCP_time_type whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.4.4).

X_status is a rank-one allocatable array of type INTEGER(ip_), that will be allocated to be of length n and filled with values that give the status of each of the bound constraints on x if record_x_status is .TRUE.. In this case, the j-th component of X_status will have one of the following values:

- 0 variable x_j lies between its bounds.
- -1 variable x_j lies on its lower bound for all feasible x (and thus may be fixed at the value X_l(j) and removed from the problem).
- 1 variable x_j lies on its upper bound for all feasible x (and thus may be fixed at the value X_u(j) and removed from the problem).
- -2 variable x_j never lies on its lower bound for any feasible x (and thus the lower bound X_l(j) may be set to minus infinity).
- 2 variable x_j never lies on its upper bound for any feasible x (and thus the upper bound X_u(j) may be set to infinity).
- 3 the bounds on variable x_j are equal (and thus x_j may be fixed at the value X_l(j) and removed from the problem).
- -3 variable x_j never lies on its either bound for any feasible x (and thus the lower bound X_l(j) may be set to minus infinity and the upper bound X_u(j) may be set to infinity).

X_status will remain unallocated if record_x_status is .FALSE..

C_status is a rank-one allocatable array of type INTEGER(ip_), that will be allocated to be of length m and filled with values that give the status of each of the general constraints on x if record_c_status is .TRUE.. In this case, the i-th component of C_status will have one of the following values:

0 constraint value $(\mathbf{Ax})_i$ lies between its bounds.

-1 constraint $(\mathbf{Ax})_i$ lies on its lower bound for all feasible x (and thus may be fixed at the value C_l(i) and treated as an equality constraint).

1 constraint value $(\mathbf{Ax})_i$ lies on its upper bound for all feasible x (and thus may be fixed at the value C_u(i) and removed from the problem).

-2 constraint value $(\mathbf{Ax})_i$ never lies on its lower bound for any feasible x (and thus the lower bound C_l(i) may be set to minus infinity).

2 constraint value $(\mathbf{Ax})_i$ never lies on its upper bound for any feasible x (and thus the upper bound C_u(i) may be set to infinity).

3 the bounds on constraint value $(\mathbf{Ax})_i$ are equal (and thus $(\mathbf{Ax})_i$ is equality constrained).

-3 constraint value $(\mathbf{Ax})_i$ never lies on its either bound for any feasible x (and thus the constraint may be removed from the problem).

4 constraint i is implied by others (and thus may be removed from the problem).

C_status will remain unallocated if record_c_status is .FALSE..

FDC_inform is a scalar variable of type FDC_inform_type whose components are used to provide information about any detection of linear dependencies performed by the package GALAHAD_FDC. See the specification sheet for the package GALAHAD_FDC for details, and appropriate default values.

SBLS_inform is a scalar variable of type SBLS_inform_type whose components are used to provide information about factorizations performed by the package GALAHAD_SBLS. See the specification sheet for the package GALAHAD_SBLS for details, and appropriate default values.

2.4.6 The derived data type for holding problem data

The derived data type WCP_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of WCP procedures. This data should be preserved, untouched, from the initial call to WCP_initialize to the final call to WCP_terminate.

2.5 Argument lists and calling sequences

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

1. The subroutine WCP_initialize is used to set default values, and initialize private data, before solving one or more problems with the same sparsity and bound structure.
2. The subroutine WCP_solve is called to solve the problem.
3. The subroutine WCP_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by WCP_solve, 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 WCP_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

2.5.1 The initialization subroutine

Default values are provided as follows:

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

data is a scalar INTENT(INOUT) argument of type WCP_data_type (see Section 2.4.6). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type WCP_control_type (see Section 2.4.3). On exit, control contains default values for the components as described in Section 2.4.3. These values should only be changed after calling WCP_initialize.

inform is a scalar INTENT(INOUT) argument of type WCP_inform_type (see Section 2.4.5). A successful call to WCP_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

2.5.2 The linear or separable convex quadratic programming problem solution subroutine

The constrained least-distance problem solution algorithm is called as follows:

```
CALL WCP_solve( p, data, control, info )
```

p is a scalar INTENT(INOUT) argument of type QPT_problem_type (see Section 2.4.2). It is used to hold data about the problem being solved. For a new problem, the user must allocate all the array components, and set values for all components except p%C. p%new_problem_structure must be set .TRUE., but will have been reset to .FALSE. on exit from WCP_solve. Users are free to choose whichever of the three matrix formats described in Section 2.1 is appropriate for **A** for their application.

For a problem with the same structure as one that has just been solved, the user may set p%new_problem_structure to .FALSE., so long as WCP_terminate has not been called in the interim. The INTEGER(ip_) components must be unaltered since the previous call to WCP_solve, but the REAL(rp_) may be altered to reflect the new problem.

The components p%X, p%Y_l, p%Y_u, p%Z_l and p%Z_u must be set to initial estimates, $\mathbf{x}^0$, of the primal variables, **x**, Lagrange multipliers for the general constraints, $\mathbf{y}_l$ and $\mathbf{y}_u$, and dual variables for the bound constraints, $\mathbf{z}_l$ and $\mathbf{z}_u$, respectively. Inappropriate initial values will be altered, so the user should not be overly concerned if suitable values are not apparent, and may be content with merely setting p%X=0.0, p%Y_l=0.0, p%Y_u=0.0, p%Z_l=0.0 and p%Z_u=0.0. The component p%C need not be set on entry.

On exit, the components p%X, p%Y_l, p%Y_u, p%Z_l, p%Z_u and p%C will contain the best estimates of the primal variables **x**, Lagrange multipliers for the general constraints $\mathbf{y}_l$ and $\mathbf{y}_u$, dual variables for the bound constraints $\mathbf{z}_l$ and $\mathbf{z}_u$, and values of the constraints **Ax** respectively. What of the remaining problem data has been restored depends upon the input value of the control parameter control%restore_problem. The return format for a restored array component will be the same as its input format. **Restrictions:** p%n > 0, p%m ≥ 0 and p%A_ne ≥ -2.

data is a scalar INTENT(INOUT) argument of type WCP_data_type (see Section 2.4.6). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to WCP_initialize.

control is a scalar INTENT(IN) argument of type WCP_control_type (see Section 2.4.3). Default values may be assigned by calling WCP_initialize prior to the first call to WCP_solve.

inform is a scalar INTENT(OUT) argument of type WCP_inform_type (see Section 2.4.5). A successful call to WCP_solve is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

2.5.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

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

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

control is a scalar INTENT(IN) argument of type WCP_control_type exactly as for WCP_solve.

inform is a scalar INTENT(OUT) argument of type WCP_inform_type exactly as for WCP_solve. Only the component status will be set on exit, and a successful call to WCP_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.6.

2.6 Warning and error messages

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

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

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

-3. One of the restrictions prob%n > 0 or prob%m ≥ 0 or requirements that prob%A_type contains its relevant string 'DENSE', 'COORDINATE' or 'SPARSE_BY_ROWS' has been violated.

-4. The equality constraints are inconsistent.

-5. The constraints appear to have no feasible point.

-10. The factorization failed; the return status from the factorization package is given in the component inform%factor_status.

-17. The problem is so ill-conditioned that further progress is impossible.

-18 Too many iterations have been performed. This may happen if control%maxit is too small, but may also be symptomatic of a badly scaled problem.

-19. The elapsed CPU or system clock time limit has been reached. This may happen if either control%cpu_time_limit or control%clock_time_limit is too small, but may also be symptomatic of a badly scaled problem.

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 WCP_control_type (see Section 2.4.3), by reading an appropriate data specification file using the subroutine WCP_read_specfile. This facility is useful as it allows a user to change WCP control parameters without editing and recompiling programs that call WCP.

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

```
( .. lines ignored by WCP_read_specfile .. )
  BEGIN WCP
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by WCP_read_specfile .. )
```

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

and

```
END WCP SPECIFICATION
```

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

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

The specification file must be open for input when WCP_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 WCP_read_specfile.

Control parameters corresponding to the components FDC_control and SBLS_control may be changed by including additional sections enclosed by "BEGIN FDC" and "END FDC", and "BEGIN SBLS" and "END SBLS", respectively. See the specification sheets for the packages GALAHAD_FDC and GALAHAD_SBLS for further details.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL WCP_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type WCP_control_type (see Section 2.4.3). Default values should have already been set, perhaps by calling WCP_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 2.4.3) of control that each affects are given in Table 2.1.

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

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
maximum-number-of-iterations	%maxit	integer
start-print	%start_print	integer
stop-print	%stop_print	integer
initial-point-used	%initial_point	integer
maximum-poor-iterations-before-infeasible	%infeas_max	integer
perturbation-strategy	%perturbation_strategy	integer
restore-problem-on-output	%restore_problem	integer
infinity-value	%infinity	real
primal-accuracy-required	%stop_p	real
dual-accuracy-required	%stop_d	real
complementary-slackness-accuracy-required	%stop_c	real
initial-bound-perturbation	perturb_start	real
perturbation-small	%perturbation_small	real
reduce-perturbation-factor	%reduce_perturb_factor	real
reduce-perturbation-multiplier	%reduce_perturb_multiplier	real
insufficiently-feasible-tolerance	%insufficiently_feasible	real
implicit-variable-tolerance	%implicit_tol	real
mininum-initial-primal-feasibility	%prfeas	real
mininum-initial-dual-feasibility	%dufeas	real
target-barrier-parameter	%mu_target	real
target-barrier-accept-fraction	%mu_accept_fraction	real
increase-barrier-parameter-by	%mu_increase_factor	real
required-infeasibility-reduction	%required_infeas_reduction	real
alpha-scaling-tolerance	%alpha_scale	real
identical-bounds-tolerance	%identical_bounds_tol	real
maximum-cpu-time-limit	%cpu_time_limit	real
maximum-clock-time-limit	%clock_time_limit	real
remove-linear-dependencies	%remove_dependencies	logical
treat-zero-bounds-as-general	%treat_zero_bounds_as_general	logical
just-find-feasible-point	%just_feasible	logical
balance-initial-complementarity	%balance_initial_complementarity	logical
use-corrector-step	%use_corrector	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
record-x-status	%record_x_status	logical
record-c-status	%record_c_status	logical

Table 2.1: Specfile commands and associated components of control.

2.8 Information printed

If control%print_level is positive, information about the progress of the algorithm will be printed on unit control%out. If control%print_level $= 1$, a single line of output will be produced for each iteration of the process. This will include values of the current primal and dual infeasibility, and violation of complementary slackness, the feasibility-phase objective value, the current steplength, the value of the barrier parameter, the number of backtracks in the linesearch and the elapsed clock time in seconds.

If control%print_level ≥ 2 this output will be increased to provide significant detail of each iteration. This extra output includes residuals of the linear systems solved, and, for larger values of control%print_level, values of the primal and dual variables and Lagrange multipliers.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: WCP_solve calls the GALAHAD packages GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SMT, GALAHAD_QPT, GALAHAD_QPD, GALAHAD_SPECFILE, GALAHAD_QPP, GALAHAD_SPACE, GALAHAD_SORT, GALAHAD_STRING, GALAHAD_ROOTS, GALAHAD_SBLS, GALAHAD_FDC.

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

Restrictions: prob%n > 0, prob%m ≥ 0, prob%A_type $\in \{$'DENSE', 'COORDINATE', 'SPARSE_BY_ROWS'$\}$.

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

4 METHOD

The algorithm is iterative, and at each major iteration attempts to find a solution to the perturbed system (1.3), (1.4),

$$(c_i - c_i^l + (\theta_c^l)_i)(y_i^l + (\theta_y^l)_i) = (\mu_c^l)_i \text{ and } (c_i - c_i^u - (\theta_c^u)_i)(y_i^u - (\theta_y^u)_i) = (\mu_c^u)_i, \quad i = 1, \ldots, m, \tag{4.1}$$

$$(x_j - x_j^l + (\theta_x^l)_j)(z_j^l + (\theta_z^l)_j) = (\mu_x^l)_j \text{ and } (x_j - x_j^u - (\theta_x^u)_j)(z_j^u - (\theta_z^u)_j) = (\mu_x^u)_j, \quad j = 1, \ldots, n, \tag{4.2}$$

and

$$\mathbf{c}^l - \boldsymbol{\theta}_c^l < \mathbf{c} < \mathbf{c}^u + \boldsymbol{\theta}_c^u, \ \mathbf{x}^l - \boldsymbol{\theta}_x^l < \mathbf{x} < \mathbf{x}^u + \boldsymbol{\theta}_x^u, \ \mathbf{y}^l > -\boldsymbol{\theta}_y^l, \ \mathbf{y}^u < \boldsymbol{\theta}_y^u, \ \mathbf{z}^l > -\boldsymbol{\theta}_z^l \text{ and } \mathbf{z}^u < \boldsymbol{\theta}_z^u, \tag{4.3}$$

where the vectors of perturbations $\boldsymbol{\theta}_c^l$, $\boldsymbol{\theta}_c^u$, $\boldsymbol{\theta}_x^l$, $\boldsymbol{\theta}_x^u$, $\boldsymbol{\theta}_x^l$, $\boldsymbol{\theta}_x^u$, $\boldsymbol{\theta}_y^l$, $\boldsymbol{\theta}_y^u$, $\boldsymbol{\theta}_z^l$ and $\boldsymbol{\theta}_z^u$, are non-negative. Rather than solve (1.3)–(1.4) and (4.1)–(4.3) exactly, we instead seek a feasible point for the easier relaxation (1.3)–(1.4) and

$$\begin{array}{rcccll} \gamma(\mu_c^l)_i & \leq & (c_i - c_i^l + (\theta_c^l)_i)(y_i^l + (\theta_y^l)_i) & \leq & (\mu_c^l)_i/\gamma & \text{and} \\ \gamma(\mu_c^u)_i & \leq & (c_i - c_i^u - (\theta_c^u)_i)(y_i^u - (\theta_y^u)_i) & \leq & (\mu_c^u)_i,/\gamma & i = 1, \ldots, m, \text{ and} \\ \gamma(\mu_x^l)_j & \leq & (x_j - x_j^l + (\theta_x^l)_j)(z_j^l + (\theta_z^l)_j) & \leq & (\mu_x^l)_j/\gamma & \text{and} \\ \gamma(\mu_x^u)_j & \leq & (x_j - x_j^u - (\theta_x^u)_j)(z_j^u - (\theta_z^u)_j) & \leq & (\mu_x^u)_j/\gamma, & j = 1, \ldots, n, \end{array} \tag{4.4}$$

for some $\gamma \in (0, 1]$ which is allowed to be smaller than one if there is a nonzero perturbation.

Given any solution to (1.3)–(1.4) and (4.4) satisfying (4.3),, the perturbations are reduced (sometimes to zero) so as to ensure that the current solution is feasible for the next perturbed problem. Specifically, the perturbation $(\theta_c^l)_i$ for the constraint $c_i \geq c_i^l$ is set to zero if c_i is larger than some given parameter $\varepsilon > 0$. If not, but c_i is strictly

positive, the perturbation will be reduced by a multiplier $\rho \in (0,1)$. Otherwise, the new perturbation will be set to $\xi(\theta_c^l)_i + (1-\xi)(c_i^l - c_i)$ for some factor $\xi \in (0,1)$. Identical rules are used to reduce the remaining primal and dual perturbations. The targets $\boldsymbol{\mu}_c^l$, $\boldsymbol{\mu}_c^u$, $\boldsymbol{\mu}_x^l$ and $\boldsymbol{\mu}_x^u$ will also be increased by the factor $\beta \geq 1$ for those (primal and/or dual) variables with strictly positive perturbations so as to try to accelerate the convergence.

Ultimately the intention is to drive all the perturbations to zero. It can be shown that if the original problem (1.3)–(1.6) and (1.8) has a solution, the perturbations will be zero after a finite number of major iterations. Equally, if there is no interior solution (1.8), the sets of (primal and dual) variables that are necessarily at (one of) their bounds for all feasible points—we refer to these as *implicit* equalities—will be identified, as will the possibility that there is no point (interior or otherwise) in the primal and/or dual feasible regions.

Each major iteration requires the solution $\mathbf{v} = (\mathbf{x}, \mathbf{c}, \mathbf{z}^l, \mathbf{z}^u, \mathbf{y}^l, \mathbf{y}^u)$ of the nonlinear system (1.3), (1.4) and (4.1)–(4.3) for fixed perturbations, using a minor iteration. The minor iteration uses a stabilized (predictor-corrector) Newton method, in which the arc $\mathbf{v}(\alpha) = \mathbf{v} + \alpha\dot{\mathbf{v}} + \alpha^2\ddot{\mathbf{v}}$, $\alpha \in [0,1]$, involving the standard Newton step $\dot{\mathbf{v}}$ for the equations (1.3), (1.4), (4.1) and (4.2), optionally augmented by a corrector $\ddot{\mathbf{v}}$ to account for the nonlinearity in (4.1)–(4.2), is truncated so as to ensure that

$$(c_i(\alpha) - c_i^l + (\theta_c^l)_i)(y_i^l(\alpha) + (\theta_y^l)_i) \geq \tau(\mu_c^l)_i \text{ and } (c_i(\alpha) - c_i^u - (\theta_c^u)_i)(y_i^u(\alpha) - (\theta_y^u)_i) \geq \tau(\mu_c^u)_i, \quad i = 1, \ldots, m,$$

and

$$(x_j(\alpha) - x_j^l + (\theta_x^l)_j)(z_j^l(\alpha) + (\theta_z^l)_j) \geq \tau(\mu_x^l)_j \text{ and } (x_j(\alpha) - x_j^u - (\theta_x^u)_j)(z_j^u(\alpha) - (\theta_z^u)_j) \geq \tau(\mu_x^u)_j, \quad j = 1, \ldots, n,$$

for some $\tau \in (0,1)$, always holds, and also so that the norm of the residuals to (1.3), (1.4), (4.1) and (4.2) is reduced as much as possible. The Newton and corrector systems are solved using a factorization of the Jacobian of its defining functions (the so-called "augmented system" approach) or of a reduced system in which some of the trivial equations are eliminated (the "Schur-complement" approach). The factors are obtained using the GALAHAD package `GALAHAD_SBLS`.

In order to make the solution as efficient as possible, the variables and constraints are reordered internally by the GALAHAD package `GALAHAD_QPP` prior to solution. In particular, fixed variables, and free (unbounded on both sides) constraints are temporarily removed. In addition, an attempt to identify and remove linearly dependent equality constraints may be made by factorizing

$$\begin{pmatrix} \alpha \mathbf{I} & \mathbf{A}_{\mathcal{E}}^T \\ \mathbf{A}_{\mathcal{E}} & \mathbf{0} \end{pmatrix},$$

where $\mathbf{A}_{\mathcal{E}}$ denotes the gradients of the equality constraints and $\alpha > 0$ is a given scaling factor, using `GALAHAD_SBLS`, and examining small pivot blocks.

References:

The basic algorithm, its convergence analysis and results of numerical experiments are given in

C. Cartis and N. I. M. Gould (2006). Finding a point n the relative interior of a polyhedron. Technical Report TR-2006-016, Rutherford Appleton Laboratory.

5 EXAMPLE OF USE

Suppose we wish to find a well-centered interior point that satisfies the general linear constraints $1 \leq 2x_1 + x_2 \leq 2$, $x_2 + x_3 = 2$, and simple bounds $-1 \leq x_1 \leq 1$ and $x_3 \leq 2$. starting from $\mathbf{x}^0 = (-2, 1, 3)^T$. Then, on writing the data for this problem as

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & \\ & 1 & 1 \end{pmatrix}, \ \mathbf{c}^l = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \ \mathbf{c}^u = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \ \mathbf{x}^l = \begin{pmatrix} -1 \\ -\infty \\ -\infty \end{pmatrix} \text{ and } \mathbf{x}^u = \begin{pmatrix} 1 \\ \infty \\ 2 \end{pmatrix},$$

we may use the following code.

```
! THIS VERSION: GALAHAD 2.4 - 04/05/2010 AT 09:00 GMT.
   PROGRAM GALAHAD_WCP_example
   USE GALAHAD_WCP_double                     ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
   REAL ( KIND = wp ), PARAMETER :: infinity = 10.0_wp ** 20
   TYPE ( QPT_problem_type ) :: p
   TYPE ( WCP_data_type ) :: data
   TYPE ( WCP_control_type ) :: control
   TYPE ( WCP_inform_type ) :: inform
   INTEGER, PARAMETER :: n = 3, m = 2, a_ne = 4
   INTEGER :: i, s
! start problem data
   ALLOCATE( p%X( n ), p%X_l( n ), p%X_u( n ), p%Z_l( n ), p%Z_u( n ) )
   ALLOCATE( p%C( m ), p%C_l( m ), p%C_u( m ), p%Y_l( m ), p%Y_u( m ) )
   p%new_problem_structure = .TRUE.           ! new structure
   p%n = n ; p%m = m ; p%f = 0.0_wp           ! dimensions & objective constant
   p%C_l = (/ 1.0_wp, 2.0_wp /)               ! constraint lower bound
   p%C_u = (/ 2.0_wp, 2.0_wp /)               ! constraint upper bound
   p%X_l = (/ - 1.0_wp, - infinity, - infinity /) ! variable lower bound
   p%X_u = (/ 1.0_wp, infinity, 2.0_wp /)     ! variable upper bound
   p%gradient_kind = 0
! sparse co-ordinate storage format: integer components
   CALL SMT_put( p%A%type, 'COORDINATE', s )     ! storage for A
   ALLOCATE( p%A%val( a_ne ), p%A%row( a_ne ), p%A%col( a_ne ) )
   p%A%row = (/ 1, 1, 2, 2 /)                 ! Jacobian A
   p%A%col = (/ 1, 2, 2, 3 /) ; p%A%ne = a_ne
! integer components complete
   CALL WCP_initialize( data, control, inform ) ! Initialize control parameters
   control%infinity = infinity                  ! Set infinity
   p%X = (/  -2.0_wp, 1.0_wp,  3.0_wp /)        ! set x0
   p%Y_l = 1.0_wp ;  p%Y_u = - 1.0_wp ; p%Z_l = 1.0_wp ;  p%Z_u = - 1.0_wp
! sparse co-ordinate storage format: real components
   p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
! real components complete
   CALL WCP_solve( p, data, control, inform )   ! Solve problem
   IF ( inform%status == 0 ) THEN               ! Successful return
     WRITE( 6, "( 1X, I0, ' iterations. objective value =', ES11.4, /,         &
    &  ' well-centered point:', /, ' i     X_l           X         X_u' )" )    &
       inform%iter, inform%obj
     DO i = 1, n
       WRITE( 6, "( I2, 3ES12.4 )" ) i, p%X_l( i ), p%X( i ), p%X_u( i )
     END DO
     WRITE( 6, "( ' constraints:', /, ' i     C_l       A * X         C_u' )" )
     DO i = 1, m
       WRITE( 6, "( I2, 3ES12.4 )" ) i, p%C_l( i ), p%C( i ), p%C_u( i )
     END DO
   ELSE                                         !  Error returns
     WRITE( 6, "( ' WCP_solve exit status = ', I6 ) " ) inform%status
   END IF
   CALL WCP_terminate( data, control, inform ) !  delete internal workspace
   END PROGRAM GALAHAD_WCP_example
```

This produces the following output:

```
 5 iterations. objective value = 0.0000E+00
 well-centered point:
 i     X_l           X         X_u
 1 -1.0000E+00 -2.3293E-01  1.0000E+00
 2 -1.0000E+20  2.0265E+00  1.0000E+20
 3 -1.0000E+20 -2.6531E-02  2.0000E+00
 constraints:
 i     C_l       A * X         C_u
 1  1.0000E+00  1.5607E+00  2.0000E+00
 2  2.0000E+00  2.0000E+00  2.0000E+00
```

The same problem may be solved holding the data in a sparse row-wise storage format by replacing the lines

```
! sparse co-ordinate storage format: integer components
...
! integer components complete
```

by

```
! sparse row-wise storage format: integer components
   CALL SMT_put( p%A%type, 'SPARSE_BY_ROWS' )  ! Specify sparse-by-row storage
   ALLOCATE( p%A%val( a_ne ), p%A%col( a_ne ), p%A%ptr( m + 1 )  )
   p%A%col = (/ 1, 2, 2, 3 /)                ! Jacobian A
   p%A%ptr = (/ 1, 3, 5 /)                   ! Set row pointers
! integer components complete
```

and

```
! sparse co-ordinate storage format: real components
...
! real components complete
```

by

```
! sparse row-wise storage format: real components
     p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
! real components complete
```

or using a dense storage format with the replacement lines

```
! dense storage format: integer components
   CALL SMT_put( p%A%type, 'DENSE' )  ! Specify dense storage for A
   ALLOCATE( p%A%val( n * m ) )
! integer components complete
```

and

```
! dense storage format: real components
     p%A%val = (/ 2.0_wp, 1.0_wp, 0.0_wp, 0.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian
! real components complete
```

respectively.