C interfaces to GALAHAD WCP

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Sun Apr 2 2023

Chapter 1

GALAHAD C package wcp

1.1 Introduction

1.1.1 Purpose

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

$$(1)\quad c_i^l \leq a_i^T x \leq c_i^u, \quad i = 1, \ldots, m,$$

and the simple bound constraints

$$(2)\quad x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n,$$

where the vectors a_i, c^l, c^u, x^l and x^u are given. More specifically, if possible, the package finds a solution to the system of primal optimality equations

$$(3)\quad Ax = c,$$

dual optimality equations

$$(4)\quad g = A^T y + z, \;\; y = y^l + y^u, \text{ and } z = z^l + z^u,$$

and perturbed complementary slackness equations

$$(5)\quad (c_i - c_i^l)y_i^l = (\mu_c^l)_i \text{ and } (c_i - c_i^u)y_i^u = (\mu_c^u)_i, \quad i = 1, \ldots, m,$$

and

$$(6)\quad ((x_j - x_j^l)z_j^l = (\mu_x^l)_j \text{ and } (x_j - x_j^u)z_j^u = (\mu_x^u)_j, \quad j = 1, \ldots, n,$$

for which

$$(7)\quad c^l \leq c \leq c^u, \;\; x^l \leq x \leq x^u, \;\; y^l \geq 0, \;\; y^u \leq 0, \;\; z^l \geq 0 \text{ and } z^u \leq 0$$

Here A is the matrix whose rows are the a_i^T, $i = 1, \ldots, m$, μ_c^l, μ_c^u, μ_x^l and μ_x^u are vectors of strictly positive {*targets*}, g is another given vector, and (y^l, y^u) and (z^l, z^u) are dual variables for the linear constraints and simple bounds respectively; c gives the constraint value Ax. Since (5)-(7) normally imply that

$$(8)\quad c^l < c < c^u, \;\; x^l < x < x^u, \;\; y^l > 0, \;\; y^u < 0, \;\; z^l > 0 \text{ and } z^u < 0$$

such a primal-dual point $(x, c, y^l, y^u, z^l, z^l)$ may be used, for example, as a feasible starting point for primal-dual interior-point methods for solving the linear programming problem of minimizing $g^T x$ subject to (1) and (2).

Full advantage is taken of any zero coefficients in the vectors 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 (8) only holds as an equality for all feasible points.

1.1.2 Authors

C. Cartis and N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England.

C interface, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

Julia interface, additionally A. Montoison and D. Orban, Polytechnique Montréal.

1.1.3 Originally released

July 2006, C interface January 2022.

1.1.4 Terminology

1.1.5 Method

The algorithm is iterative, and at each major iteration attempts to find a solution to the perturbed system (3), (4),

$$(9)\quad (c_i - c_i^l + (\theta_c^l)_i)(y_i^l + (\theta_y^l)_i) = (\mu_c^l)_i \text{ and } (c_i - c_i^u - (\theta_c^u)_i)(y_i^u - (\theta_y^u)_i) = (\mu_c^u)_i, \quad i = 1, \ldots, m,$$

$$(10)\quad (x_j - x_j^l + (\theta_x^l)_j)(z_j^l + (\theta_z^l)_j) = (\mu_x^l)_j \text{ and } (x_j - x_j^u - (\theta_x^u)_j)(z_j^u - (\theta_z^u)_j) = (\mu_x^u)_j, \quad j = 1, \ldots, n,$$

and

$$(11)\quad c^l - \theta_c^l < c < c^u + \theta_c^u, \;\; x^l - \theta_x^l < x < x^u + \theta_x^u, \;\; y^l > -\theta_y^l, \;\; y^u < \theta_y^u, \;\; z^l > -\theta_z^l \text{ and } z^u < \theta_z^u,$$

where the vectors of perturbations θ_c^l, θ_c^u, θ_x^l, θ_x^u, θ_x^l, θ_x^u, θ_y^l, θ_y^u, θ_z^l and θ_z^u, are non-negative. Rather than solve (3)-(4) and (9)-(11) exactly, we instead seek a feasible point for the easier relaxation (3)-(4) and

$$(12)\quad \begin{array}{rcccll} \gamma(\mu_c^l)_i & \leq & (c_i - c_i^l + (\theta_c^l)_i)(y_i^l + (\theta_y^l)_i) & \leq & (\mu_c^l)_i/\gamma & \text{and} \\ \gamma(\mu_c^u)_i & \leq & (c_i - c_i^u - (\theta_c^u)_i)(y_i^u - (\theta_y^u)_i) & \leq & (\mu_c^u)_i, /\gamma & i = 1, \ldots, m, \text{ and} \\ \gamma(\mu_x^l)_j & \leq & (x_j - x_j^l + (\theta_x^l)_j)(z_j^l + (\theta_z^l)_j) & \leq & (\mu_x^l)_j/\gamma & \text{and} \\ \gamma(\mu_x^u)_j & \leq & (x_j - x_j^u - (\theta_x^u)_j)(z_j^u - (\theta_z^u)_j) & \leq & (\mu_x^u)_j/\gamma, & j = 1, \ldots, n, \end{array}$$

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

Given any solution to (3)-(4) and (12) satisfying (11), 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 $\epsilon > 0$. 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 μ_c^l, μ_c^u, μ_x^l and μ_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 (3)-(6) and (8) has a solution, the perturbations will be zero after a finite number of major iterations. Equally, if there is no interior solution (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 $u = (x, c, z^l, z^u, y^l, y^u)$ of the nonlinear system (3), (4) and (9)-(11) for fixed perturbations, using a minor iteration. The minor iteration uses a stabilized (predictor-corrector) Newton method, in which the arc $u(\alpha) = u + \alpha\dot{u} + \alpha^2\ddot{u}, \alpha \in [0, 1]$, involving the standard Newton step $\dot{u}$ for the equations

(3), (4), (9) and (10), optionally augmented by a corrector $\bar{u}$ account for the nonlinearity in (9) and (10), 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 (3), (4), (9) and (10) 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 SBLS.

In order to make the solution as efficient as possible, the variables and constraints are reordered internally by the GALAHAD package 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

$$\begin{pmatrix} \alpha I & A_E^T \\ A_E & 0 \end{pmatrix},$$

where A_E denotes the gradients of the equality constraints and $\alpha > 0$ is a given scaling factor, using the GALAHAD package SBLS, and examining small pivot blocks.

1.1.6 Reference

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.

1.1.7 Call order

To solve a given problem, functions from the wcp package must be called in the following order:

- wcp_initialize - provide default control parameters and set up initial data structures
- wcp_read_specfile (optional) - override control values by reading replacement values from a file
- wcp_import - set up problem data structures and fixed values
- wcp_reset_control (optional) - possibly change control parameters if a sequence of problems are being solved
- wcp_find_wcp - find a well-centered point
- wcp_information (optional) - recover information about the solution and solution process
- wcp_terminate - deallocate data structures

See Section 4.1 for examples of use.

1.1.8 Unsymmetric matrix storage formats

The unsymmetric m by n constraint matrix A may be presented and stored in a variety of convenient input formats.

Both C-style (0 based) and fortran-style (1-based) indexing is allowed. Choose `control.f_indexing` as `false` for C style and `true` for fortran style; the discussion below presumes C style, but add 1 to indices for the corresponding fortran version.

Wrappers will automatically convert between 0-based (C) and 1-based (fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing.

1.1.8.1 Dense storage format

The matrix A 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. In this case, component $n * i + j$ of the storage array A_val will hold the value A_{ij} for $0 \leq i \leq m - 1, 0 \leq j \leq n - 1$.

1.1.8.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry, $0 \leq l \leq ne - 1$, of A, its row index i, column index j and value A_{ij}, $0 \leq i \leq m - 1$, $0 \leq j \leq n - 1$, are stored as the l-th components of the integer arrays A_row and A_col and real array A_val, respectively, while the number of nonzeros is recorded as A_ne = ne.

1.1.8.3 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 A the i-th component of the integer array A_ptr holds the position of the first entry in this row, while A_ptr(m) holds the total number of entries. The column indices j, $0 \leq j \leq n - 1$, and values A_{ij} of the nonzero entries in the i-th row are stored in components l = A_ptr(i), . . ., A_ptr(i+1)-1, $0 \leq i \leq m - 1$, 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.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 galahad_wcp.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_fdc.h"
#include "galahad_sbls.h"
```

Data Structures

- struct wcp_control_type
- struct wcp_time_type
- struct wcp_inform_type

Functions

- void wcp_initialize (void **data, struct wcp_control_type *control, int *status)
- void wcp_read_specfile (struct wcp_control_type *control, const char specfile[ ])
- void wcp_import (struct wcp_control_type *control, void **data, int *status, int n, int m, const char A_type[ ], int A_ne, const int A_row[ ], const int A_col[ ], const int A_ptr[ ])
- void wcp_reset_control (struct wcp_control_type *control, void **data, int *status)
- void wcp_find_wcp (void **data, int *status, int n, int m, const real_wp_ g[ ], int a_ne, const real_wp_ A_↩ val[ ], const real_wp_ c_l[ ], const real_wp_ c_u[ ], const real_wp_ x_l[ ], const real_wp_ x_u[ ], real_wp_ x[ ], real_wp_ c[ ], real_wp_ y_l[ ], real_wp_ y_u[ ], real_wp_ z_l[ ], real_wp_ z_u[ ], int x_stat[ ], int c_stat[ ])
- void wcp_information (void **data, struct wcp_inform_type *inform, int *status)
- void wcp_terminate (void **data, struct wcp_control_type *control, struct wcp_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct wcp_control_type

control derived type as a C struct

Examples

wcpt.c, and wcptf.c.

Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	error and warning diagnostics occur on stream error
int	out	general output occurs on stream out
int	print_level	the level of output required is specified by print_level
int	start_print	any printing will start on this iteration
int	stop_print	any printing will stop on this iteration
int	maxit	at most maxit inner iterations are allowed
int	initial_point	how to choose the initial point. Possible values are • 0 the values input in X, shifted to be at least prfeas from their nearest bound, will be used • 1 the nearest point to the "bound average" 0.5(X_l+X_u) that satisfies the linear constraints will be used
int	factor	the factorization to be used. Possible values are • 0 automatic • 1 Schur-complement factorization • 2 augmented-system factorization
int	max_col	the maximum number of nonzeros in a column of A which is permitted with the Schur-complement factorization
int	indmin	an initial guess as to the integer workspace required by SBLS
int	valmin	an initial guess as to the real workspace required by SBLS
int	itref_max	the maximum number of iterative refinements allowed
int	infeas_max	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_reducti

Data Fields

int	perturbation_strategy	the strategy used to reduce relaxed constraint bounds. Possible values are • 0 do not perturb the constraints • 1 reduce all perturbations by the same amount with linear reduction • 2 reduce each perturbation as much as possible with linear reduction • 3 reduce all perturbations by the same amount with superlinear reduction • 4 reduce each perturbation as much as possible with superlinear reduction
int	restore_problem	indicate whether and how much of the input problem should be restored on output. Possible values are • 0 nothing restored • 1 scalar and vector parameters • 2 all parameters
real_wp_	infinity	any bound larger than infinity in modulus will be regarded as infinite
real_wp_	stop_p	the required accuracy for the primal infeasibility
real_wp_	stop_d	the required accuracy for the dual infeasibility
real_wp_	stop_c	the required accuracy for the complementarity
real_wp_	prfeas	initial primal variables will not be closer than prfeas from their bound
real_wp_	dufeas	initial dual variables will not be closer than dufeas from their bounds
real_wp_	mu_target	the target value of the barrier parameter. If mu_target is not positive, it will be reset to an appropriate value
real_wp_	mu_accept_fraction	the complemtary slackness x_i.z_i will be judged to lie within an acceptable margin around its target value mu as soon as mu_accept_fraction $*$ mu <= x_i.z_i <= (1 / mu_accept_fraction) $*$ mu; the perturbations will be reduced as soon as all of the complemtary slacknesses x_i.z_i lie within acceptable bounds. mu_accept_fraction will be reset to ensure that it lies in the interval (0,1]
real_wp_	mu_increase_factor	the target value of the barrier parameter will be increased by mu_increase_factor for infeasible constraints every time the perturbations are adjusted
real_wp_	required_infeas_reduction	if the overall infeasibility of the problem is not reduced by at least a factor required_infeas_reduction over .infeas_max iterations, the problem is flagged as infeasible (see infeas_max)

Data Fields

real_wp_	implicit_tol	any primal or dual variable that is less feasible than implicit_tol will be regarded as defining an implicit constraint
real_wp_	pivot_tol	the threshold pivot used by the matrix factorization. See the documentation for SBLS for details (obsolete)
real_wp_	pivot_tol_for_dependencies	the threshold pivot used by the matrix factorization when attempting to detect linearly dependent constraints. See the documentation for SBLS for details (obsolete)
real_wp_	zero_pivot	any pivots smaller than zero_pivot in absolute value will be regarded to zero when attempting to detect linearly dependent constraints (obsolete)
real_wp_	perturb_start	the constraint bounds will initially be relaxed by .perturb_start; this perturbation will subsequently be reduced to zero. If perturb_start < 0, the amount by which the bounds are relaxed will be computed automatically
real_wp_	alpha_scale	the test for rank defficiency will be to factorize (alpha_scale I A^T) (A 0)
real_wp_	identical_bounds_tol	any pair of constraint bounds (c_l,c_u) or (x_l,x_u) that are closer tha identical_bounds_tol will be reset to the average of their values
real_wp_	reduce_perturb_factor	the constraint perturbation will be reduced as follows: • - if the variable lies outside a bound, the corresponding perturbation will be reduced to reduce_perturb_factor $*$ current pertubation – (1 - reduce_perturb_factor) $*$ violation • - otherwise, if the variable lies within insufficiently_feasible of its bound the pertubation will be reduced to reduce_perturb_multiplier $*$ current pertubation • - otherwise if will be set to zero
real_wp_	reduce_perturb_multiplier	see reduce_perturb_factor
real_wp_	insufficiently_feasible	see reduce_perturb_factor
real_wp_	perturbation_small	if the maximum constraint pertubation is smaller than perturbation_small and the violation is smaller than implicit_tol, the method will deduce that there is a feasible point but no interior
real_wp_	cpu_time_limit	the maximum CPU time allowed (-ve means infinite)
real_wp_	clock_time_limit	the maximum elapsed clock time allowed (-ve means infinite)

Data Fields

bool	remove_dependencies	the equality constraints will be preprocessed to remove any linear dependencies if true
bool	treat_zero_bounds_as_general	any problem bound with the value zero will be treated as if it were a general value if true
bool	just_feasible	if .just_feasible is true, the algorithm will stop as soon as a feasible point is found. Otherwise, the optimal solution to the problem will be found
bool	balance_initial_complementarity	if .balance_initial_complementarity is .true. the initial complemetarity will be balanced
bool	use_corrector	if .use_corrector, a corrector step will be used
bool	space_critical	if .space_critical true, every effort will be made to use as little space as possible. This may result in longer computation time
bool	deallocate_error_fatal	if .deallocate_error_fatal is true, any array/pointer deallocation error will terminate execution. Otherwise, computation will continue
bool	record_x_status	if .record_x_status is true, the array inform.X_status will be allocated and the status of the bound constraints will be reported on exit.
bool	record_c_status	if .record_c_status is true, the array inform.C_status will be allocated and the status of the general constraints will be reported on exit.
char	prefix[31]	all output lines will be prefixed by .prefix(2:LEN(TRIM(.prefix))-1) where .prefix contains the required string enclosed in quotes, e.g. "string" or 'string'
struct fdc_control_type	fdc_control	control parameters for FDC
struct sbls_control_type	sbls_control	control parameters for SBLS

3.1.1.2 struct wcp_time_type

time derived type as a C struct

Data Fields

real_wp_	total	the total CPU time spent in the package
real_wp_	preprocess	the CPU time spent preprocessing the problem
real_wp_	find_dependent	the CPU time spent detecting linear dependencies
real_wp_	analyse	the CPU time spent analysing the required matrices prior to factorization
real_wp_	factorize	the CPU time spent factorizing the required matrices
real_wp_	solve	the CPU time spent computing the search direction
real_wp_	clock_total	the total clock time spent in the package
real_wp_	clock_preprocess	the clock time spent preprocessing the problem
real_wp_	clock_find_dependent	the clock time spent detecting linear dependencies
real_wp_	clock_analyse	the clock time spent analysing the required matrices prior to factorization
real_wp_	clock_factorize	the clock time spent factorizing the required matrices
real_wp_	clock_solve	the clock time spent computing the search direction

3.1.1.3 struct wcp_inform_type

inform derived type as a C struct

Examples

wcpt.c, and wcptf.c.

Data Fields

int	status	return status. See WCP_solve for details
int	alloc_status	the status of the last attempted allocation/deallocation
char	bad_alloc[81]	the name of the array for which an allocation/deallocation error ocurred
int	iter	the total number of iterations required
int	factorization_status	the return status from the factorization
int64_t	factorization_integer	the total integer workspace required for the factorization
int64_t	factorization_real	the total real workspace required for the factorization
int	nfacts	the total number of factorizations performed
int	c_implicit	the number of general constraints that lie on (one) of their bounds for feasible solutions
int	x_implicit	the number of variables that lie on (one) of their bounds for all feasible solutions
int	y_implicit	the number of Lagrange multipliers for general constraints that lie on (one) of their bounds for all feasible solutions
int	z_implicit	the number of dual variables that lie on (one) of their bounds for all feasible solutions
real_wp_	obj	the value of the objective function at the best estimate of the solution determined by WCP_solve
real_wp_	mu_final_target_max	the largest target value on termination
real_wp_	non_negligible_pivot	the smallest pivot which was not judged to be zero when detecting linear dependent constraints
bool	feasible	is the returned "solution" feasible?
struct wcp_time_type	time	timings (see above)
struct fdc_inform_type	fdc_inform	inform parameters for FDC
struct sbls_inform_type	sbls_inform	inform parameters for SBLS

3.1.2 Function Documentation

3.1.2.1 wcp_initialize()

```
void wcp_initialize (
             void ** data,
             struct wcp_control_type * control,
             int * status )
```

Set default control values and initialize private data

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see wcp_control_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The import was succesful.

Examples

wcpt.c, and wcptf.c.

3.1.2.2 wcp_read_specfile()

```
void wcp_read_specfile (
             struct wcp_control_type * control,
             const char specfile[] )
```

Read the content of a specification file, and assign values associated with given keywords to the corresponding control parameters. By default, the spcification file will be named RUNWCP.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in $GALAHAD/doc/wcp.pdf for a list of keywords that may be set.

Parameters

in,out	*control*	is a struct containing control information (see wcp_control_type)
in	*specfile*	is a character string containing the name of the specification file

3.1.2.3 wcp_import()

```
void wcp_import (
             struct wcp_control_type * control,
             void ** data,
             int * status,
             int n,
             int m,
             const char A_type[],
             int A_ne,
             const int A_row[],
             const int A_col[],
             const int A_ptr[] )
```

Import problem data into internal storage prior to solution.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see wcp_control_type)
in,out	*data*	holds private internal data
in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The import was succesful • -1. An allocation error occurred. A message indicating the offending array is written on unit control.error, and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -2. A deallocation error occurred. A message indicating the offending array is written on unit control.error and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -3. The restrictions n $>$ 0 or m $>$ 0 or requirement that a type contains its relevant string 'dense', 'coordinate', 'sparse_by_rows', 'diagonal', 'scaled_identity', 'identity', 'zero' or 'none' has been violated.
in	*n*	is a scalar variable of type int, that holds the number of variables.
in	*m*	is a scalar variable of type int, that holds the number of general linear constraints.
in	*A_type*	is a one-dimensional array of type char that specifies the unsymmetric storage scheme used for the constraint Jacobian, A. It should be one of 'coordinate', 'sparse_by_rows' or 'dense; lower or upper case variants are allowed.
in	*A_ne*	is a scalar variable of type int, that holds the number of entries in A in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	*A_row*	is a one-dimensional array of size A_ne and type int, that holds the row indices of A in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes, and in this case can be NULL.
in	*A_col*	is a one-dimensional array of size A_ne and type int, that holds the column indices of A in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense or diagonal storage schemes are used, and in this case can be NULL.
in	*A_ptr*	is a one-dimensional array of size n+1 and type int, that holds the starting position of each row of A, as well as the total number of entries, in the sparse row-wise storage scheme. It need not be set when the other schemes are used, and in this case can be NULL.

Examples

wcpt.c, and wcptf.c.

3.1.2.4 wcp_reset_control()

```
void wcp_reset_control (
             struct wcp_control_type * control,
             void ** data,
             int * status )
```

Reset control parameters after import if required.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see wcp_control_type)
in,out	*data*	holds private internal data
in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The import was succesful.

3.1.2.5 wcp_find_wcp()

```
void wcp_find_wcp (
             void ** data,
             int * status,
             int n,
             int m,
             const real_wp_ g[],
             int a_ne,
             const real_wp_ A_val[],
             const real_wp_ c_l[],
             const real_wp_ c_u[],
             const real_wp_ x_l[],
             const real_wp_ x_u[],
             real_wp_ x[],
             real_wp_ c[],
             real_wp_ y_l[],
             real_wp_ y_u[],
             real_wp_ z_l[],
             real_wp_ z_u[],
             int x_stat[],
             int c_stat[] )
```

Find a well-centered point in the feasible region

Parameters

in,out	*data*	holds private internal data

Parameters

in,out	*status*	is a scalar variable of type int, that gives the entry and exit status from the package. Possible exit are: • 0. The run was succesful • -1. An allocation error occurred. A message indicating the offending array is written on unit control.error, and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -2. A deallocation error occurred. A message indicating the offending array is written on unit control.error and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -3. The restrictions n $>$ 0 and m $>$ 0 or requirement that a type contains its relevant string 'dense', 'coordinate', 'sparse_by_rows', 'diagonal', 'scaled_identity', 'identity', 'zero' or 'none' has been violated. • -4. The constraint bounds are inconsistent. • -5. The constraints appear to have no feasible point. • -9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component inform.factor_status • -10. The factorization failed; the return status from the factorization package is given in the component inform.factor_status. • -11. The solution of a set of linear equations using factors from the factorization package failed; the return status from the factorization package is given in the component inform.factor_status. • -16. The problem is so ill-conditioned that further progress is impossible. • -17. The step is too small to make further impact. • -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 CPU time limit has been reached. This may happen if control.cpu_time_limit is too small, but may also be symptomatic of a badly scaled problem.
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*m*	is a scalar variable of type int, that holds the number of general linear constraints.
in	*g*	is a one-dimensional array of size n and type double, that holds the vector g. The j-th component of g, j = 0, ... , n-1, contains g_j.
in	*a_ne*	is a scalar variable of type int, that holds the number of entries in the constraint Jacobian matrix A.
in	*A_val*	is a one-dimensional array of size a_ne and type double, that holds the values of the entries of the constraint Jacobian matrix A in any of the available storage schemes.
in	*c_l*	is a one-dimensional array of size m and type double, that holds the lower bounds c^l on the constraints Ax. The i-th component of c_l, i = 0, ... , m-1, contains c^l_i.
in	*c_u*	is a one-dimensional array of size m and type double, that holds the upper bounds c^l on the constraints Ax. The i-th component of c_u, i = 0, ... , m-1, contains c^u_i.
in	*x_l*	is a one-dimensional array of size n and type double, that holds the lower bounds x^l on the variables x. The j-th component of x_l, j = 0, ... , n-1, contains x^l_j.

Parameters

in	*x_u*	is a one-dimensional array of size n and type double, that holds the upper bounds x^l on the variables x. The j-th component of x_u, j = 0, ... , n-1, contains x_j^l.
in,out	*x*	is a one-dimensional array of size n and type double, that holds the values x of the optimization variables. The j-th component of x, j = 0, ... , n-1, contains x_j.
out	*c*	is a one-dimensional array of size m and type double, that holds the residual $c(x)$. The i-th component of c, i = 0, ... , m-1, contains $c_i(x)$.
in,out	*y_l*	is a one-dimensional array of size n and type double, that holds the values y^l of the Lagrange multipliers for the lower bounds on the general linear constraints. The j-th component of y_l, i = 0, ... , m-1, contains y_i^l.
in,out	*y_u*	is a one-dimensional array of size n and type double, that holds the values y^u of the Lagrange multipliers for the upper bounds on the general linear constraints. The j-th component of y_u, i = 0, ... , m-1, contains y_i^u.
in,out	*z_l*	is a one-dimensional array of size n and type double, that holds the values z^l of the dual variables for the lower bounds on the variables. The j-th component of z_l, j = 0, ... , n-1, contains z_j^l.
in,out	*z_u*	is a one-dimensional array of size n and type double, that holds the values z^u of the dual variables for the upper bounds on the variables. The j-th component of z_u, j = 0, ... , n-1, contains z_j^u.
out	*x_stat*	is a one-dimensional array of size n and type int, that gives the optimal status of the problem variables. If x_stat(j) is negative, the variable x_j most likely lies on its lower bound, if it is positive, it lies on its upper bound, and if it is zero, it lies between its bounds.
out	*c_stat*	is a one-dimensional array of size m and type int, that gives the optimal status of the general linear constraints. If c_stat(i) is negative, the constraint value $a_i^T x$ most likely lies on its lower bound, if it is positive, it lies on its upper bound, and if it is zero, it lies between its bounds.

Examples

wcpt.c, and wcptf.c.

3.1.2.6 wcp_information()

```
void wcp_information (
            void ** data,
            struct wcp_inform_type * inform,
            int * status )
```

Provides output information.

Parameters

in,out	*data*	holds private internal data
out	*inform*	is a struct containing output information (see wcp_inform_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The values were recorded succesfully

Examples

wcpt.c, and wcptf.c.

3.1.2.7 wcp_terminate()

```
void wcp_terminate (
            void ** data,
            struct wcp_control_type * control,
            struct wcp_inform_type * inform )
```

Deallocate all internal private storage.

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see wcp_control_type)
out	*inform*	is a struct containing output information (see wcp_inform_type)

Examples

wcpt.c, and wcptf.c.

Chapter 4

Example Documentation

4.1 wcpt.c

This is an example of how to use the package to solve a quadratic program. A variety of supported Hessian and constraint matrix storage formats are shown.

Notice that C-style indexing is used, and that this is flaggeed by setting `control.f_indexing` to `false`.

```
/* wcpf.c */
/* Full test for the WCP C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_wcp.h"
int main(void) {
    // Derived types
    void *data;
    struct wcp_control_type control;
    struct wcp_inform_type inform;
    // Set problem data
    int n = 3; // dimension
    int m = 2; // number of general constraints
    real_wp_ g[] = {0.0, 2.0, 0.0};   // linear term in the objective
    int A_ne = 4; // Jacobian elements
    int A_row[] = {0, 0, 1, 1}; // row indices
    int A_col[] = {0, 1, 1, 2}; // column indices
    int A_ptr[] = {0, 2, 4}; // row pointers
    real_wp_ A_val[] = {2.0, 1.0, 1.0, 1.0 }; // values
    real_wp_ c_l[] = {1.0, 2.0};   // constraint lower bound
    real_wp_ c_u[] = {2.0, 2.0};   // constraint upper bound
    real_wp_ x_l[] = {-1.0, - INFINITY, - INFINITY}; // variable lower bound
    real_wp_ x_u[] = {1.0, INFINITY, 2.0}; // variable upper bound
    // Set output storage
    real_wp_ c[m]; // constraint values
    int x_stat[n]; // variable status
    int c_stat[m]; // constraint status
    char st;
    int status;
    printf(" C sparse matrix indexing\n\n");
    printf(" basic tests of wcp storage formats\n\n");
    for( int d=1; d <= 3; d++){
        // Initialize WCP
        wcp_initialize( &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = false; // C sparse matrix indexing
        // Start from 0
        real_wp_ x[] = {0.0,0.0,0.0};
        real_wp_ y_l[] = {0.0,0.0};
        real_wp_ y_u[] = {0.0,0.0};
        real_wp_ z_l[] = {0.0,0.0,0.0};
        real_wp_ z_u[] = {0.0,0.0,0.0};
        switch(d){
            case 1: // sparse co-ordinate storage
                st = 'C';
```

```
                wcp_import( &control, &data, &status, n, m,
                            "coordinate", A_ne, A_row, A_col, NULL );
                wcp_find_wcp( &data, &status, n, m, g, A_ne, A_val,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            printf(" case %1i break\n",d);
            case 2: // sparse by rows
                st = 'R';
                wcp_import( &control, &data, &status, n, m,
                             "sparse_by_rows", A_ne, NULL, A_col, A_ptr );
                wcp_find_wcp( &data, &status, n, m, g, A_ne, A_val,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            case 3: // dense
                st = 'D';
                int A_dense_ne = 6; // number of elements of A
                real_wp_ A_dense[] = {2.0, 1.0, 0.0, 0.0, 1.0, 1.0};
                wcp_import( &control, &data, &status, n, m,
                             "dense", A_dense_ne, NULL, NULL, NULL );
                wcp_find_wcp( &data, &status, n, m, g, A_dense_ne, A_dense,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            }
        wcp_information( &data, &inform, &status );
        if(inform.status == 0){
            printf("%c:%6i iterations. Optimal objective value = %5.2f status = %1i\n",
                   st, inform.iter, inform.obj, inform.status);
        }else{
            printf("%c: WCP_solve exit status = %1i\n", st, inform.status);
        }
        //printf("x: ");
        //for( int i = 0; i < n; i++) printf("%f ", x[i]);
        //printf("\n");
        //printf("gradient: ");
        //for( int i = 0; i < n; i++) printf("%f ", g[i]);
        //printf("\n");
        // Delete internal workspace
        wcp_terminate( &data, &control, &inform );
    }
}
```

4.2 wcptf.c

This is the same example, but now fortran-style indexing is used.

```
/* wcptf.c */
/* Full test for the WCP C interface using Fortran sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_wcp.h"
int main(void) {
    // Derived types
    void *data;
    struct wcp_control_type control;
    struct wcp_inform_type inform;
    // Set problem data
    int n = 3; // dimension
    int m = 2; // number of general constraints
    real_wp_ g[] = {0.0, 2.0, 0.0};   // linear term in the objective
    int A_ne = 4; // Jacobian elements
    int A_row[] = {1, 1, 2, 2}; // row indices
    int A_col[] = {1, 2, 2, 3}; // column indices
    int A_ptr[] = {1, 3, 5}; // row pointers
    real_wp_ A_val[] = {2.0, 1.0, 1.0, 1.0 }; // values
    real_wp_ c_l[] = {1.0, 2.0};   // constraint lower bound
    real_wp_ c_u[] = {2.0, 2.0};   // constraint upper bound
    real_wp_ x_l[] = {-1.0, - INFINITY, - INFINITY}; // variable lower bound
    real_wp_ x_u[] = {1.0, INFINITY, 2.0}; // variable upper bound
    // Set output storage
    real_wp_ c[m]; // constraint values
    int x_stat[n]; // variable status
    int c_stat[m]; // constraint status
    char st;
    int status;
    printf(" Fortran sparse matrix indexing\n\n");
    printf(" basic tests of wcp storage formats\n\n");
```

```
    for( int d=1; d <= 3; d++){
        // Initialize WCP
        wcp_initialize( &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = true; // Fortran sparse matrix indexing
        // Start from 0
        real_wp_ x[] = {0.0,0.0,0.0};
        real_wp_ y_l[] = {0.0,0.0};
        real_wp_ y_u[] = {0.0,0.0};
        real_wp_ z_l[] = {0.0,0.0,0.0};
        real_wp_ z_u[] = {0.0,0.0,0.0};
        switch(d){
            case 1: // sparse co-ordinate storage
                st = 'C';
                wcp_import( &control, &data, &status, n, m,
                            "coordinate", A_ne, A_row, A_col, NULL );
                wcp_find_wcp( &data, &status, n, m, g, A_ne, A_val,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            printf(" case %1i break\n",d);
            case 2: // sparse by rows
                st = 'R';
                wcp_import( &control, &data, &status, n, m,
                             "sparse_by_rows", A_ne, NULL, A_col, A_ptr );
                wcp_find_wcp( &data, &status, n, m, g, A_ne, A_val,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            case 3: // dense
                st = 'D';
                int A_dense_ne = 6; // number of elements of A
                real_wp_ A_dense[] = {2.0, 1.0, 0.0, 0.0, 1.0, 1.0};
                wcp_import( &control, &data, &status, n, m,
                            "dense", A_dense_ne, NULL, NULL, NULL );
                wcp_find_wcp( &data, &status, n, m, g, A_dense_ne, A_dense,
                              c_l, c_u, x_l, x_u, x, c, y_l, y_u, z_l, z_u,
                              x_stat, c_stat );
                break;
            }
        wcp_information( &data, &inform, &status );
        if(inform.status == 0){
            printf("%c:%6i iterations. Optimal objective value = %5.2f status = %1i\n",
                   st, inform.iter, inform.obj, inform.status);
        }else{
            printf("%c: WCP_solve exit status = %1i\n", st, inform.status);
        }
        //printf("x: ");
        //for( int i = 0; i < n; i++) printf("%f ", x[i]);
        //printf("\n");
        //printf("gradient: ");
        //for( int i = 0; i < n; i++) printf("%f ", g[i]);
        //printf("\n");
        // Delete internal workspace
        wcp_terminate( &data, &control, &inform );
    }
}
```