C interfaces to GALAHAD DPS

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Tue May 2 2023

Chapter 1

GALAHAD C package dps

1.1 Introduction

1.1.1 Purpose

Given a real n by n symmetric matrix H, this package **construct a symmetric, positive definite matrix M so that H is diagonal in the norm $\|v\|_M = \sqrt{v^T M v}$ induced by M.** Subsequently the package can be use to **solve the trust-region subproblem**

$$\text{(1) minimize } q(x) = \frac{1}{2} x^T H x + c^T x + f \text{ subject to } \|x\|\|_M \leq \Delta$$

or the **regularized quadratic problem**

$$\text{(2) minimize } q(x) + \frac{1}{p} \sigma \|x\|\|_M^p$$

for a real n vector c and scalars f, $\Delta > 0$, $\sigma > 0$ and $p \geq 2$.

A factorization of the matrix H will be required, so this package is most suited for the case where such a factorization, either dense or sparse, may be found efficiently.

1.1.2 Authors

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

August 2011, C interface December 2021.

1.1.4 Terminology

1.1.5 Method

The required solution x_* necessarily satisfies the optimality condition $Hx_* + \lambda_* M x_* + c = 0$, where $\lambda_* \geq 0$ is a Lagrange multiplier that corresponds to the constraint $\|x\|_M \leq \Delta$ in the trust-region case (1), and is given by $\lambda_* = \sigma \|x_*\|^{p-2}$ for the regularization problem (2). In addition $H + \lambda_* M$ will be positive semi-definite; in most instances it will actually be positive definite, but in special `‘hard'' cases singularity is a possibility.

The matrix H is decomposed as

$$H = PLDL^T P^T$$

by calling the GALAHAD package SLS. Here P is a permutation matrix, L is unit lower triangular and D is block diagonal, with blocks of dimension at most two. The spectral decomposition of each diagonal block of D is computed, and each eigenvalue θ is replaced by $\max(|\theta|, \theta_{\min})$, where $\theta_{\min}$ is a positive user-supplied value. The resulting block diagonal matrix is B, from which we define the **modified-absolute-value**

$$M = PLBL^T P^T;$$

an alternative due to Goldfarb uses instead the simpler

$$M = PLL^T P^T.$$

Given the factors of H (and M), the required solution is found by making the change of variables $y = B^{1/2} L^T P^T x$ (or $y = L^T P^T x$ in the Goldfarb case) which results in `‘diagonal'' trust-region and regularization subproblems, whose solution may be easily obtained suing a Newton or higher-order iteration of a resulting `‘secular'' equation. If subsequent problems, for which H and c are unchanged, are to be attempted, the existing factorization and solution may easily be exploited.

The dominant cost is that for the factorization of the symmetric, but potentially indefinite, matrix H using the GALAHAD package SLS.

1.1.6 Reference

The method is described in detail for the trust-region case in

N. I. M. Gould and J. Nocedal (1998). The modified absolute-value factorization for trust-region minimization. In `‘High Performance Algorithms and Software in Nonlinear Optimization'' (R. De Leone, A. Murli, P. M. Pardalos and G. Toraldo, eds.), Kluwer Academic Publishers, pp. 225-241,

while the adaptation for the regularization case is obvious. The method used to solve the diagonal trust-region and regularization subproblems are as given by

H. S. Dollar, N. I. M. Gould and D. P. Robinson (2010). On solving trust-region and other regularised subproblems in optimization. Mathematical Programming Computation **2(1)** 21-57

with simplifications due to the diagonal Hessian.

1.1.7 Call order

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

- dps_initialize - provide default control parameters and set up initial data structures
- dps_read_specfile (optional) - override control values by reading replacement values from a file
- dps_import - import control and matrix data structures
- dps_reset_control (optional) - possibly change control parameters if a sequence of problems are being solved
- one of
  - dps_solve_tr_problem - solve the trust-region problem (1)
  - dps_solve_rq_problem - solve the regularized-quadratic problem (2)
- optionally one of
  - dps_resolve_tr_problem - resolve the trust-region problem (1) when the non-matrix data has changed
  - dps_resolve_rq_problem - resolve the regularized-quadratic problem (2) when the non-matrix data has changed
- dps_information (optional) - recover information about the solution and solution process
- dps_terminate - deallocate data structures

See Section **??** for examples of use.

1.1.8 Symmetric matrix storage formats

The symmetric n by n coefficient matrix H may be presented and stored in a variety of convenient input formats. Crucially symmetry is exploited by only storing values from the lower triangular part (i.e, those entries that lie on or below the leading diagonal).

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 H 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. Since H is symmetric, only the lower triangular part (that is the part H_{ij} for $0 \leq j \leq i \leq n-1$) need be held. In this case the lower triangle should be stored by rows, that is component $i * i/2 + j$ of the storage array val will hold the value H_{ij} (and, by symmetry, H_{ji}) for $0 \leq j \leq i \leq n-1$.

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 H, its row index i, column index j and value H_{ij}, $0 \leq j \leq i \leq n-1$, are stored as the l-th components of the integer arrays row and col and real array val, respectively, while the number of nonzeros is recorded as ne = ne. Note that only the entries in the lower triangle should be stored.

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 H the i-th component of the integer array ptr holds the position of the first entry in this row, while ptr(n) holds the total number of entries. The column indices j, $0 \leq j \leq i$, and values H_{ij} of the entries in the i-th row are stored in components l = ptr(i), . . ., ptr(i+1)-1 of the integer array col, and real array val, respectively. Note that as before only the entries in the lower triangle should be stored. 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:

galahad_dps.h . . . . . ??

Chapter 3

File Documentation

3.1 galahad_dps.h File Reference

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

Data Structures

- struct dps_control_type
- struct dps_time_type
- struct dps_inform_type

Functions

- void dps_initialize (void **data, struct dps_control_type *control, int *status)
- void dps_read_specfile (struct dps_control_type *control, const char specfile[ ])
- void dps_import (struct dps_control_type *control, void **data, int *status, int n, const char H_type[ ], int ne, const int H_row[ ], const int H_col[ ], const int H_ptr[ ])
- void dps_reset_control (struct dps_control_type *control, void **data, int *status)
- void dps_solve_tr_problem (void **data, int *status, int n, int ne, real_wp_ H_val[ ], real_wp_ c[ ], real_wp_ f, real_wp_ radius, real_wp_ x[ ])
- void dps_solve_rq_problem (void **data, int *status, int n, int ne, real_wp_ H_val[ ], real_wp_ c[ ], real_wp_ f, real_wp_ power, real_wp_ weight, real_wp_ x[ ])
- void dps_resolve_tr_problem (void **data, int *status, int n, real_wp_ c[ ], real_wp_ f, real_wp_ radius, real_wp_ x[ ])
- void dps_resolve_rq_problem (void **data, int *status, int n, real_wp_ c[ ], real_wp_ f, real_wp_ power, real_wp_ weight, real_wp_ x[ ])
- void dps_information (void **data, struct dps_inform_type *inform, int *status)
- void dps_terminate (void **data, struct dps_control_type *control, struct dps_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct dps_control_type

control derived type as a C struct

Examples

dpst.c.

Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	unit for error messages
int	out	unit for monitor output
int	problem	unit to write problem data into file problem_file
int	print_level	controls level of diagnostic output
int	new_h	how much of H has changed since the previous call. Possible values are • 0 unchanged • 1 values but not indices have changed • 2 values and indices have changed
int	taylor_max_degree	maximum degree of Taylor approximant allowed
real_wp_	eigen_min	smallest allowable value of an eigenvalue of the block diagonal factor of H
real_wp_	lower	lower and upper bounds on the multiplier, if known
real_wp_	upper	see lower
real_wp_	stop_normal	stop trust-region solution when $\|\|\|x\|\|_M - \delta\| \leq$ max(.stop_normal $*$ delta, .stop_absolute_normal)
real_wp_	stop_absolute_normal	see stop_normal
bool	goldfarb	use the Goldfarb variant of the trust-region/regularization norm rather than the modified absolute-value version
bool	space_critical	if space is critical, ensure allocated arrays are no bigger than needed
bool	deallocate_error_fatal	exit if any deallocation fails
char	problem_file[31]	name of file into which to write problem data
char	symmetric_linear_solver[31]	symmetric (indefinite) linear equation solver
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 sls_control_type	sls_control	control parameters for the Cholesky factorization and solution

3.1.1.2 struct dps_time_type

time derived type as a C struct

Data Fields

real_wp_	total	total CPU time spent in the package
real_wp_	analyse	CPU time spent reordering H prior to factorization.
real_wp_	factorize	CPU time spent factorizing H.
real_wp_	solve	CPU time spent solving the diagonal model system.
real_wp_	clock_total	total clock time spent in the package
real_wp_	clock_analyse	clock time spent reordering H prior to factorization
real_wp_	clock_factorize	clock time spent factorizing H
real_wp_	clock_solve	clock time spent solving the diagonal model system

3.1.1.3 struct dps_inform_type

inform derived type as a C struct

Examples

dpst.c.

Data Fields

int	status	return status. See DPS_solve for details
int	alloc_status	STAT value after allocate failure.
int	mod_1by1	the number of 1 by 1 blocks from the factorization of H that were modified when constructing M
int	mod_2by2	the number of 2 by 2 blocks from the factorization of H that were modified when constructing M
real_wp_	obj	the value of the quadratic function
real_wp_	obj_regularized	the value of the regularized quadratic function
real_wp_	x_norm	the M-norm of the solution
real_wp_	multiplier	the Lagrange multiplier associated with the constraint/regularization
real_wp_	pole	a lower bound max(0,-lambda_1), where lambda_1 is the left-most eigenvalue of (H, M)
bool	hard_case	has the hard case occurred?
char	bad_alloc[81]	name of array that provoked an allocate failure
struct dps_time_type	time	time information
struct sls_inform_type	sls_inform	information from SLS

3.1.2 Function Documentation

3.1.2.1 dps_initialize()

```
void dps_initialize (
            void ** data,
```

```
            struct dps_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 dps_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

dpst.c.

3.1.2.2 dps_read_specfile()

```
void dps_read_specfile (
            struct dps_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 RUNDPS.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in $GALAHAD/doc/dps.pdf for a list of keywords that may be set.

Parameters

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

3.1.2.3 dps_import()

```
void dps_import (
            struct dps_control_type * control,
            void ** data,
            int * status,
            int n,
            const char H_type[],
            int ne,
            const int H_row[],
            const int H_col[],
            const int H_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 dps_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: • 1. The import was succesful, and the package is ready for the solve phase • -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 restriction n $>$ 0 or requirement that type contains its relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated.
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*H_type*	is a one-dimensional array of type char that specifies the symmetric storage scheme used for the Hessian. It should be one of 'coordinate', 'sparse_by_rows' or 'dense'; lower or upper case variants are allowed
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of H in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	*H_row*	is a one-dimensional array of size ne and type int, that holds the row indices of the lower triangular part of H in the sparse co-ordinate storage scheme. It need not be set for any of the other three schemes, and in this case can be NULL
in	*H_col*	is a one-dimensional array of size ne and type int, that holds the column indices of the lower triangular part of H 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	*H_ptr*	is a one-dimensional array of size n+1 and type int, that holds the starting position of each row of the lower triangular part of H, 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

dpst.c.

3.1.2.4 dps_reset_control()

```
void dps_reset_control (
             struct dps_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 dps_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: • 1. The import was succesful, and the package is ready for the solve phase

3.1.2.5 dps_solve_tr_problem()

```
void dps_solve_tr_problem (
            void ** data,
            int * status,
            int n,
            int ne,
            real_wp_ H_val[],
            real_wp_ c[],
            real_wp_ f,
            real_wp_ radius,
            real_wp_ x[] )
```

Find the global minimizer of the trust-region problem (1).

Parameters

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 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 restriction n $>$ 0 or requirement that type contains its relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated. • -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. • -16. The problem is so ill-conditioned that further progress is impossible. • -40. An error has occured when building the preconditioner.

Parameters

in	*n*	is a scalar variable of type int, that holds the number of variables
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of the Hessian matrix H.
in	*H_val*	is a one-dimensional array of size ne and type double, that holds the values of the entries of the lower triangular part of the Hessian matrix H in any of the available storage schemes.
in	*c*	is a one-dimensional array of size n and type double, that holds the linear term c in the objective function. The j-th component of c, j = 0, ... , n-1, contains c_j.
in	*f*	is a scalar variable pointer of type double, that holds the value of the holds the constant term f in the objective function.
in	*radius*	is a scalar variable pointer of type double, that holds the value of the trust-region radius, $\Delta > 0$.
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.

Examples

dpst.c.

3.1.2.6 dps_solve_rq_problem()

```
void dps_solve_rq_problem (
             void ** data,
             int * status,
             int n,
             int ne,
             real_wp_ H_val[],
             real_wp_ c[],
             real_wp_ f,
             real_wp_ power,
             real_wp_ weight,
             real_wp_ x[] )
```

Find the global minimizer of the regularized-quadartic problem (2).

Parameters

in,out	*data*	holds private internal data

Parameters

in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values 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 restriction n $>$ 0 or requirement that type contains its relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated. • -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. • -16. The problem is so ill-conditioned that further progress is impossible. • -40. An error has occured when building the preconditioner.
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of the Hessian matrix H.
in	*H_val*	is a one-dimensional array of size ne and type double, that holds the values of the entries of the lower triangular part of the Hessian matrix H in any of the available storage schemes.
in	*c*	is a one-dimensional array of size n and type double, that holds the linear term c in the objective function. The j-th component of c, j = 0, ... , n-1, contains c_j.
in	*f*	is a scalar variable pointer of type double, that holds the value of the holds the constant term f in the objective function.
in	*weight*	is a scalar variable pointer of type double, that holds the value of the regularization weight, $\sigma > 0$.
in	*power*	is a scalar variable pointer of type double, that holds the value of the regularization power, $p \geq 2$.
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.

3.1.2.7 dps_resolve_tr_problem()

```
void dps_resolve_tr_problem (
             void ** data,
             int * status,
             int n,
             real_wp_ c[],
```

```
            real_wp_ f,
            real_wp_ radius,
            real_wp_ x[] )
```

Find the global minimizer of the trust-region problem (1) if some non-matrix components have changed since a call to dps_solve_tr_problem.

Parameters

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 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 restriction n $>$ 0 or requirement that type contains its relevant string 'dense', 'coordinate' or 'sparse_by_rows' has been violated. • -16. The problem is so ill-conditioned that further progress is impossible.
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*c*	is a one-dimensional array of size n and type double, that holds the linear term c in the objective function. The j-th component of c, j = 0, ... , n-1, contains c_j.
in	*f*	is a scalar variable pointer of type double, that holds the value of the constant term f in the objective function.
in	*radius*	is a scalar variable pointer of type double, that holds the value of the trust-region radius, $\Delta > 0$.
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.

Examples

dpst.c.

3.1.2.8 dps_resolve_rq_problem()

```
void dps_resolve_rq_problem (
            void ** data,
            int * status,
            int n,
            real_wp_ c[],
            real_wp_ f,
            real_wp_ power,
```

```
            real_wp_ weight,
            real_wp_ x[] )
```

Find the global minimizer of the regularized-quadartic problem (2) if some non-matrix components have changed since a call to dps_solve_rq_problem.

Parameters

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 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. • -16. The problem is so ill-conditioned that further progress is impossible.
in	*n*	is a scalar variable of type int, that holds the number of variables
in	*c*	is a one-dimensional array of size n and type double, that holds the linear term c in the objective function. The j-th component of c, j = 0, ... , n-1, contains c_j.
in	*f*	is a scalar variable pointer of type double, that holds the value of the holds the constant term f in the objective function.
in	*weight*	is a scalar variable pointer of type double, that holds the value of the regularization weight, $\sigma > 0$.
in	*power*	is a scalar variable pointer of type double, that holds the value of the regularization power, $p \geq 2$.
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.

3.1.2.9 dps_information()

```
void dps_information (
            void ** data,
            struct dps_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 dps_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

dpst.c.

3.1.2.10 dps_terminate()

```
void dps_terminate (
            void ** data,
            struct dps_control_type * control,
            struct dps_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 dps_control_type)
out	*inform*	is a struct containing output information (see dps_inform_type)

Examples

dpst.c.

Chapter 4

Example Documentation

4.1 dpst.c

This is an example of how to use the package.

```
/* dpst.c */
/* Full test for the DPS C interface using C sparse matrix indexing */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_dps.h"
int main(void) {
    // Derived types
    void *data;
    struct dps_control_type control;
    struct dps_inform_type inform;
    // Set problem data
    int n = 3; // dimension of H
    int m = 1; // dimension of A
    int H_ne = 4; // number of elements of H
    int H_dense_ne = 6; // number of elements of H
    int H_row[] = {0, 1, 2, 2}; // row indices, NB lower triangle
    int H_col[] = {0, 1, 2, 0};
    int H_ptr[] = {0, 1, 2, 4};
    real_wp_ H_val[] = {1.0, 2.0, 3.0, 4.0};
    real_wp_ H_dense[] = {1.0, 0.0, 2.0, 4.0, 0.0, 3.0};
    real_wp_ f = 0.96;
    real_wp_ radius = 1.0;
    real_wp_ half_radius = 0.5;
    real_wp_ c[] = {0.0, 2.0, 0.0};
    char st;
    int status;
    real_wp_ x[n];
    printf(" C sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    for( int storage_type=1; storage_type <= 3; storage_type++){
    // Initialize DPS
      dps_initialize( &data, &control, &status );
    // Set user-defined control options
      control.f_indexing = false; // C sparse matrix indexing
      strcpy(control.symmetric_linear_solver,"sytr ") ;
      switch(storage_type){
        case 1: // sparse co-ordinate storage
            st = 'C';
            // import the control parameters and structural data
            dps_import( &control, &data, &status, n,
                        "coordinate", H_ne, H_row, H_col, NULL );
            // solve the problem
            dps_solve_tr_problem( &data, &status, n, H_ne, H_val,
                                  c, f, radius, x );
            break;
        case 2: // sparse by rows
            st = 'R';
            // import the control parameters and structural data
            dps_import( &control, &data, &status, n,
                        "sparse_by_rows", H_ne, NULL, H_col, H_ptr );
```

```
            dps_solve_tr_problem( &data, &status, n, H_ne, H_val,
                                  c, f, radius, x );
            break;
          case 3: // dense
            st = 'D';
            // import the control parameters and structural data
            dps_import( &control, &data, &status, n,
                        "dense", H_ne, NULL, NULL, NULL );
            dps_solve_tr_problem( &data, &status, n, H_dense_ne, H_dense,
                                  c, f, radius, x );
            break;
        }
        dps_information( &data, &inform, &status );
        printf("format %c: DPS_solve_problem exit status   = %1i, f = %.2f\n",
               st, inform.status, inform.obj );
        switch(storage_type){
          case 1: // sparse co-ordinate storage
            st = 'C';
            // solve the problem
            dps_resolve_tr_problem( &data, &status, n,
                                    c, f, half_radius, x );
            break;
          case 2: // sparse by rows
            st = 'R';
            dps_resolve_tr_problem( &data, &status, n,
                                    c, f, half_radius, x );
            break;
          case 3: // dense
            st = 'D';
            dps_resolve_tr_problem( &data, &status, n,
                                    c, f, half_radius, x );
            break;
        }
        dps_information( &data, &inform, &status );
        printf("format %c: DPS_resolve_problem exit status = %1i, f = %.2f\n",
               st, inform.status, inform.obj );
        //printf("x: ");
        //for( int i = 0; i < n+m; i++) printf("%f ", x[i]);
        // Delete internal workspace
        dps_terminate( &data, &control, &inform );
    }
}
```