C interfaces to GALAHAD PSLS

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Mon May 1 2023

Chapter 1

GALAHAD C package psls

1.1 Introduction

1.1.1 Purpose

Given an n by n sparse symmetric matrix $A = a_{ij}$, this package **builds a suitable symmetric, positive definite (or diagonally dominant)-preconditioner P of A or a symmetric sub-matrix thereof**. The matrix A need not be definite. Facilities are provided to apply the preconditioner to a given vector, and to remove rows and columns (symmetrically) from the initial preconditioner without a full re-factorization.

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

April 2008, C interface January 2022.

1.1.4 Method and references

The basic preconditioners are described in detail in

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1992). LANCELOT. A fortran package for large-scale nonlinear optimization (release A). Springer Verlag Series in Computational Mathematics 17, Berlin, Section 3.3.10,

along with the more modern versions implements in ICFS due to

C.-J. Lin and J. J. More' (1999). Incomplete Cholesky factorizations with limited memory. SIAM Journal on Scientific Computing **21** 21-45,

and in HSL_MI28 described by

J. A. Scott and M. Tuma (2013). HSL MI28: an efficient and robust limited-memory incomplete Cholesky factorization code. ACM Transactions on Mathematical Software **40(4)** (2014), Article 24.

The factorization methods used by the GALAHAD package SLS in conjunction with some preconditioners are described in the documentation to that package. The key features of the external solvers supported by SLS are given in the following table.

Table 1.1 External solver characteristics

solver	factorization	indefinite A	out-of-core	parallelised
SILS/MA27	multifrontal	yes	no	no
HSL_MA57	multifrontal	yes	no	no
HSL_MA77	multifrontal	yes	yes	OpenMP core
HSL_MA86	left-looking	yes	no	OpenMP fully
HSL_MA87	left-looking	no	no	OpenMP fully
HSL_MA97	multifrontal	yes	no	OpenMP core
SSIDS	multifrontal	yes	no	CUDA core
MUMPS	multifrontal	yes	optionally	MPI
PARDISO	left-right-looking	yes	no	OpenMP fully
MKL_PARDISO	left-right-looking	yes	optionally	OpenMP fully
PaStix	left-right-looking	yes	no	OpenMP fully
WSMP	left-right-looking	yes	no	OpenMP fully
POTR	dense	no	no	with parallel LAPACK
SYTR	dense	yes	no	with parallel LAPACK
PBTR	dense band	no	no	with parallel LAPACK

Note that **the solvers themselves do not form part of this package and must be obtained separately.** Dummy instances are provided for solvers that are unavailable.

Orderings to reduce the bandwidth, as implemented in HSL's MC61, are due to

J. K. Reid and J. A. Scott (1999) Ordering symmetric sparse matrices for small profile and wavefront International Journal for Numerical Methods in Engineering **45** 1737-1755.

If a subset of the rows and columns are specified, the remaining rows/columns are removed before processing. Any subsequent removal of rows and columns is achieved using the GALAHAD Schur-complement updating package SCU unless a complete re-factorization is likely more efficient.

1.1.5 Call order

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

- psls_initialize - provide default control parameters and set up initial data structures
- psls_read_specfile (optional) - override control values by reading replacement values from a file
- psls_import - set up matrix data structures for A prior to solution
- psls_reset_control (optional) - possibly change control parameters if a sequence of problems are being solved
- one of
  - psls_form_preconditioner - form and factorize a preconditioner P of the matrix A
  - psls_form_subset_preconditioner - form and factorize a preconditioner P of a symmetric submatrix of the matrix A
- psls_update_preconditioner (optional) - update the preconditioner P when rows (amd columns) are removed
- psls_apply_preconditioner - solve the linear system of equations $Px = b$
- psls_information (optional) - recover information about the preconditioner and solution process
- psls_terminate - deallocate data structures

See Section 4.1 for examples of use.

1.1.6 Symmetric matrix storage formats

The symmetric n by n coefficient matrix A 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.

1.1.6.1 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. Since A is symmetric, only the lower triangular part (that is the part A_{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 A_{ij} (and, by symmetry, A_{ji}) for $0 \leq j \leq i \leq n-1$.

1.1.6.2 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}, $0 \leq j \leq i \leq n-1$, 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.

1.1.6.3 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 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 A_{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:

Chapter 3

File Documentation

3.1 galahad_psls.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sls.h"
#include "hsl_mi28.h"
```

Data Structures

- struct psls_control_type
- struct psls_time_type
- struct psls_inform_type

Functions

- void psls_initialize (void **data, struct psls_control_type *control, int *status)
- void psls_read_specfile (struct psls_control_type *control, const char specfile[ ])
- void psls_import (struct psls_control_type *control, void **data, int *status, int n, const char type[ ], int ne, const int row[ ], const int col[ ], const int ptr[ ])
- void psls_reset_control (struct psls_control_type *control, void **data, int *status)
- void psls_form_preconditioner (void **data, int *status, int ne, const real_wp_ val[ ])
- void psls_form_subset_preconditioner (void **data, int *status, int ne, const real_wp_ val[ ], int n_sub, const int sub[ ])
- void psls_update_preconditioner (void **data, int *status, int ne, const real_wp_ val[ ], int n_del, const int del[ ])
- void psls_apply_preconditioner (void **data, int *status, int n, real_wp_ sol[ ])
- void psls_information (void **data, struct psls_inform_type *inform, int *status)
- void psls_terminate (void **data, struct psls_control_type *control, struct psls_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct psls_control_type

control derived type as a C struct

Examples

pslst.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	print_level	controls level of diagnostic output
int	preconditioner	which preconditioner to use: • $<$0 no preconditioning occurs, $P = I$ • 0 the preconditioner is chosen automatically (forthcoming, and currently defaults to 1). • 1 A is replaced by the diagonal, P = diag(max(A, .min_diagonal)). • 2 A is replaced by the band P = band(A) with semi-bandwidth .semi_bandwidth. • 3 A is replaced by the reordered band P = band(order(A)) with semi-bandwidth .semi_bandwidth, where order is chosen by the HSL package MC61 to move entries closer to the diagonal. • 4 P is a full factorization of A using Schnabel-Eskow modifications, in which small or negative diagonals are made sensibly positive during the factorization. • 5 P is a full factorization of A due to Gill, Murray, Ponceleon and Saunders, in which an indefinite factorization is altered to give a positive definite one. • 6 P is an incomplete Cholesky factorization of A using the package ICFS due to Lin and More'. • 7 P is an incomplete factorization of A implemented as HSL_MI28 from HSL. • 8 P is an incomplete factorization of A due to Munskgaard (forthcoming). • $>$8 treated as 0. **N.B.** Options 3-8 may require additional external software that is not part of the package, and that must be obtained separately.
int	semi_bandwidth	the semi-bandwidth for band(H) when .preconditioner = 2,3
int	scaling	not used at present
int	ordering	see scaling
int	max_col	maximum number of nonzeros in a column of A for Schur-complement factorization to accommodate newly deleted rpws and columns
int	icfs_vectors	number of extra vectors of length n required by the Lin-More' incomplete Cholesky preconditioner when .preconditioner = 6

Data Fields

int	mi28_lsize	the maximum number of fill entries within each column of the incomplete factor L computed by HSL_MI28 when .preconditioner = 7. In general, increasing mi28_lsize improve the quality of the preconditioner but increases the time to compute and then apply the preconditioner. Values less than 0 are treated as 0
int	mi28_rsize	the maximum number of entries within each column of the strictly lower triangular matrix R used in the computation of the preconditioner by HSL_MI28 when .preconditioner = 7. Rank-1 arrays of size .mi28_rsize $*$ n are allocated internally to hold R. Thus the amount of memory used, as well as the amount of work involved in computing the preconditioner, depends on mi28_rsize. Setting mi28_rsize $>$ 0 generally leads to a higher quality preconditioner than using mi28_rsize = 0, and choosing mi28_rsize >= mi28_lsize is generally recommended
real_wp_	min_diagonal	the minimum permitted diagonal in diag(max(H,.min_diagonal))
bool	new_structure	set new_structure true if the storage structure for the input matrix has changed, and false if only the values have changed
bool	get_semi_bandwidth	set get_semi_bandwidth true if the semi-bandwidth of the submatrix is to be calculated
bool	get_norm_residual	set get_norm_residual true if the residual when applying the preconditioner are to be calculated
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	definite_linear_solver[31]	the definite linear equation solver used when .preconditioner = 3,4. Possible choices are currently: sils, ma27, ma57, ma77, ma86, ma87, ma97, ssids, mumps, pardiso, mkl_pardiso,pastix, wsmp, potr and pbtr, although only sils, potr, pbtr and, for OMP 4.0-compliant compilers, ssids are installed by default.
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 SLS
struct mi28_control	mi28_control	control parameters for HSL_MI28

3.1.1.2 struct psls_time_type

time derived type as a C struct

Data Fields

real_sp_	total	total time
real_sp_	assemble	time to assemble the preconditioner prior to factorization
real_sp_	analyse	time for the analysis phase
real_sp_	factorize	time for the factorization phase

Data Fields

real_sp_	solve	time for the linear solution phase
real_sp_	update	time to update the factorization
real_wp_	clock_total	total clock time spent in the package
real_wp_	clock_assemble	clock time to assemble the preconditioner prior to factorization
real_wp_	clock_analyse	clock time for the analysis phase
real_wp_	clock_factorize	clock time for the factorization phase
real_wp_	clock_solve	clock time for the linear solution phase
real_wp_	clock_update	clock time to update the factorization

3.1.1.3 struct psls_inform_type

inform derived type as a C struct

Examples

pslst.c.

Data Fields

int	status	reported return status: • 0 success • -1 allocation error • -2 deallocation error • -3 matrix data faulty (.n $<$ 1, .ne $<$ 0)
int	alloc_status	STAT value after allocate failure.
int	analyse_status	status return from factorization
int	factorize_status	status return from factorization
int	solve_status	status return from solution phase
int64_t	factorization_integer	number of integer words to hold factors
int64_t	factorization_real	number of real words to hold factors
int	preconditioner	code for the actual preconditioner used (see control.preconditioner)
int	semi_bandwidth	the actual semi-bandwidth
int	reordered_semi_bandwidth	the semi-bandwidth following reordering (if any)
int	out_of_range	number of indices out-of-range
int	duplicates	number of duplicates
int	upper	number of entries from the strict upper triangle
int	missing_diagonals	number of missing diagonal entries for an allegedly-definite matrix
int	semi_bandwidth_used	the semi-bandwidth used
int	neg1	number of 1 by 1 pivots in the factorization
int	neg2	number of 2 by 2 pivots in the factorization
bool	perturbed	has the preconditioner been perturbed during the fctorization?
real_wp_	fill_in_ratio	ratio of fill in to original nonzeros

Data Fields

real_wp_	norm_residual	the norm of the solution residual
char	bad_alloc[81]	name of array which provoked an allocate failure
int	mc61_info[10]	the integer and real output arrays from mc61
real_wp_	mc61_rinfo[15]	see mc61_info
struct psls_time_type	time	times for various stages
struct sls_inform_type	sls_inform	inform values from SLS
struct mi28_info	mi28_info	the output structure from mi28

3.1.2 Function Documentation

3.1.2.1 psls_initialize()

```
void psls_initialize (
            void ** data,
            struct psls_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 psls_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

pslst.c.

3.1.2.2 psls_read_specfile()

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

Parameters

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

3.1.2.3 psls_import()

```
void psls_import (
            struct psls_control_type * control,
            void ** data,
            int * status,
            int n,
            const char type[],
            int ne,
            const int row[],
            const int col[],
            const int ptr[] )
```

Import structural matrix data into internal storage prior to solution.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see psls_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', 'sparse_by_rows' or 'diagonal' has been violated.
in	*n*	is a scalar variable of type int, that holds the number of rows in the symmetric matrix A.
in	*type*	is a one-dimensional array of type char that specifies the symmetric storage scheme used for the matrix A. 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 A in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	*row*	is a one-dimensional array of size ne and type int, that holds the row indices of the lower triangular part of A 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.

Parameters

in	*col*	is a one-dimensional array of size ne and type int, that holds the column indices of the lower triangular part of A in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense storage scheme is used, and in this case can be NULL.
in	*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 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

pslst.c.

3.1.2.4 psls_reset_control()

```
void psls_reset_control (
             struct psls_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 psls_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 psls_form_preconditioner()

```
void psls_form_preconditioner (
             void ** data,
             int * status,
             int ne,
             const real_wp_ val[] )
```

Form and factorize a preconditioner P of the matrix A.

Parameters

in,out	*data*	holds private internal data

Parameters

out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The factors were generated succesfully. • -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. • -26. The requested solver is not available. • -29. This option is not available with this solver.
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of the symmetric matrix A.
in	*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 symmetric matrix A in any of the supported storage schemes.

Examples

pslst.c.

3.1.2.6 psls_form_subset_preconditioner()

```
void psls_form_subset_preconditioner (
            void ** data,
            int * status,
            int ne,
            const real_wp_ val[],
            int n_sub,
            const int sub[] )
```

Form and factorize a P preconditioner of a symmetric submatrix of the matrix A.

Parameters

in,out	*data*	holds private internal data

Parameters

out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The factors were generated succesfully. • -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. • -26. The requested solver is not available. • -29. This option is not available with this solver.
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of the symmetric matrix A.
in	*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 symmetric matrix A in any of the supported storage schemes.
in	*n_sub*	is a scalar variable of type int, that holds the number of rows (and columns) of the required submatrix of A.
in	*sub*	is a one-dimensional array of size n_sub and type int, that holds the indices of the rows of required submatrix.

3.1.2.7 psls_update_preconditioner()

```
void psls_update_preconditioner (
             void ** data,
             int * status,
             int ne,
             const real_wp_ val[],
             int n_del,
             const int del[] )
```

Update the preconditioner P when rows (amd columns) are removed.

Parameters

in,out	*data*	holds private internal data

Parameters

out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The factors were generated succesfully. • -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. • -26. The requested solver is not available. • -29. This option is not available with this solver.
in	*ne*	is a scalar variable of type int, that holds the number of entries in the lower triangular part of the symmetric matrix A.
in	*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 symmetric matrix A in any of the supported storage schemes.
in	*n_del*	is a scalar variable of type int, that holds the number of rows (and columns) of (sub) matrix that are to be deleted.
in	*del*	is a one-dimensional array of size n_fix and type int, that holds the indices of the rows that are to be deleted.

3.1.2.8 psls_apply_preconditioner()

```
void psls_apply_preconditioner (
             void ** data,
             int * status,
             int n,
             real_wp_ sol[] )
```

Solve the linear system $Px = b$.

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 required solution was obtained. • -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.

Parameters

in	*n*	is a scalar variable of type int, that holds the number of entries in the vectors b and x.
in,out	*sol*	is a one-dimensional array of size n and type double. On entry, it must hold the vector b. On a successful exit, its contains the solution x. Any component corresponding to rows/columns not in the initial subset recorded by psls_form_subset_preconditioner, or in those subsequently deleted by psls_update_preconditioner, will not be altered.

Examples

pslst.c.

3.1.2.9 psls_information()

```
void psls_information (
            void ** data,
            struct psls_inform_type * inform,
            int * status )
```

Provide output information

Parameters

in,out	*data*	holds private internal data
out	*inform*	is a struct containing output information (see psls_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

pslst.c.

3.1.2.10 psls_terminate()

```
void psls_terminate (
            void ** data,
            struct psls_control_type * control,
            struct psls_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 psls_control_type)
out	*inform*	is a struct containing output information (see psls_inform_type)

Examples

pslst.c.

Chapter 4

Example Documentation

4.1 pslst.c

This is an example of how to use the package.

```
/* pslst.c */
/* Full test for the PSLS C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_psls.h"
int main(void) {
    // Derived types
    void *data;
    struct psls_control_type control;
    struct psls_inform_type inform;
    // Set problem data
    int n = 5; // dimension of A
    int ne = 7; // number of elements of A
    int dense_ne = n * ( n + 1 ) / 2; // number of elements of dense A
    int row[] = {0, 1, 1, 2, 2, 3, 4}; // A indices & values, NB lower triangle
    int col[] = {0, 0, 4, 1, 2, 2, 4};
    int ptr[] = {0, 1, 3, 5, 6, 7};
    real_wp_ val[] = {2.0, 3.0, 6.0, 4.0, 1.0, 5.0, 1.0};
    real_wp_ dense[] = {2.0, 3.0, 0.0, 0.0, 4.0, 1.0, 0.0,
                        0.0, 5.0, 0.0, 0.0, 6.0, 0.0, 0.0, 1.0};
    char st;
    int status;
    int status_apply;
    printf(" C sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    for( int d=1; d <= 3; d++){
        // Initialize PSLS
        psls_initialize( &data, &control, &status );
        control.preconditioner = 2; // band preconditioner
        control.semi_bandwidth = 1; // semibandwidth
        strcpy( control.definite_linear_solver, "sils" );
        // Set user-defined control options
        control.f_indexing = false; // C sparse matrix indexing
        switch(d){
            case 1: // sparse co-ordinate storage
                st = 'C';
                psls_import( &control, &data, &status, n,
                             "coordinate", ne, row, col, NULL );
                psls_form_preconditioner( &data, &status, ne, val );
                break;
            printf(" case %1i break\n",d);
            case 2: // sparse by rows
                st = 'R';
                psls_import( &control, &data, &status, n,
                             "sparse_by_rows", ne, NULL, col, ptr );
                psls_form_preconditioner( &data, &status, ne, val );
                break;
            case 3: // dense
                st = 'D';
                psls_import( &control, &data, &status, n,
```

```
                          "dense", ne, NULL, NULL, NULL );
                psls_form_preconditioner( &data, &status, dense_ne, dense );
                break;
            }
        // Set right-hand side b in x
        real_wp_ x[] = {8.0, 45.0, 31.0, 15.0, 17.0};   // values
        if(status == 0){
          psls_information( &data, &inform, &status );
          psls_apply_preconditioner( &data, &status_apply, n, x );
        }else{
          status_apply = - 1;
        }
        printf("%c storage: status from form & factorize = %i apply = %i\n",
                   st, status, status_apply );
        //printf("x: ");
        //for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // Delete internal workspace
        psls_terminate( &data, &control, &inform );
    }
}
```