C interfaces to GALAHAD ULS

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory

Sat Mar 26 2022

Chapter 1

GALAHAD C package uls

1.1 Introduction

1.1.1 Purpose

This package **solves dense or sparse unsymmetric systems of linear equations** using variants of Gaussian elimination. Given a sparse symmetric $m \times n$ matrix $A = a_{ij}$, and an m-vector b, this subroutine solves the system $Ax = b$. If b is an n-vector, the package may solve instead the system $A^T x = b$. Both square ($m = n$) and rectangular ($m \neq n$) matrices are handled; one of an infinite class of solutions for consistent systems will be returned whenever A is not of full rank.

The method provides a common interface to a variety of well-known solvers from HSL. Currently supported solvers include `MA28/GLS` and `HSL_MA48`. 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. Also note that additional flexibility may be obtained by calling the solvers directly rather that via this package.

1.1.2 Authors

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

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

1.1.3 Originally released

August 2009, C interface December 2021.

1.1.4 Terminology

The solvers used each produce an $P_R L U P_C$ factorization of A, where L and U are lower and upper triangular matrices, and P_R and P_C are row and column permutation matrices respectively.

1.1.5 Method

Variants of sparse Gaussian elimination are used.

The solver GLS is available as part of GALAHAD and relies on the HSL Archive packages MA33. To obtain HSL Archive packages, see

http://hsl.rl.ac.uk/archive/ .

The solver HSL_MA48 is part of HSL 2007. To obtain HSL 2007 packages, see

http://hsl.rl.ac.uk/hsl2007/ .

1.1.6 Reference

The methods used are described in the user-documentation for

HSL 2007, A collection of {F}ortran codes for large-scale scientific computation (2007).
http://www.cse.clrc.ac.uk/nag/hsl

1.1.7 Call order

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

- uls_initialize - provide default control parameters and set up initial data structures
- uls_read_specfile (optional) - override control values by reading replacement values from a file
- uls_factorize_matrix - set up matrix data structures, analyse the structure to choose a suitable order for factorization, and then factorize the matrix A
- uls_reset_control (optional) - possibly change control parameters if a sequence of problems are being solved
- uls_solve_system - solve the linear system of equations $Ax = b$ or $A^T x = b$
- uls_information (optional) - recover information about the solution and solution process
- uls_terminate - deallocate data structures

See Section 4.1 for examples of use.

1.1.8 Unsymmetric matrix storage formats

The unsymmetric m by n 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 plus one. 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), $\ldots$, 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_uls.h File Reference

```
#include <stdbool.h>
#include "galahad_precision.h"
#include "galahad_gls.h"
#include "hsl_ma48.h"
```

Data Structures

- struct uls_control_type
- struct uls_inform_type

Functions

- void uls_initialize (const char solver[ ], void ∗∗data, struct uls_control_type ∗control, int ∗status)
- void uls_read_specfile (struct uls_control_type ∗control, const char specfile[ ])
- void uls_factorize_matrix (struct uls_control_type ∗control, void ∗∗data, int ∗status, int m, int n, const char type[ ], int ne, const real_wp_ val[ ], const int row[ ], const int col[ ], const int ptr[ ])
- void uls_reset_control (struct uls_control_type ∗control, void ∗∗data, int ∗status)
- void uls_solve_system (void ∗∗data, int ∗status, int m, int n, real_wp_ sol[ ], bool trans)
- void uls_information (void ∗∗data, struct uls_inform_type ∗inform, int ∗status)
- void uls_terminate (void ∗∗data, struct uls_control_type ∗control, struct uls_inform_type ∗inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct uls_control_type

control derived type as a C struct

Examples

ulst.c, and ulstf.c.

Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	unit for error messages
int	warning	unit for warning messages
int	out	unit for monitor output
int	print_level	controls level of diagnostic output
int	print_level_solver	controls level of diagnostic output from external solver
int	initial_fill_in_factor	prediction of factor by which the fill-in will exceed the initial number of nonzeros in A
int	min_real_factor_size	initial size for real array for the factors and other data
int	min_integer_factor_size	initial size for integer array for the factors and other data
int	max_factor_size	maximum size for real array for the factors and other data
int	blas_block_size_factorize	level 3 blocking in factorize
int	blas_block_size_solve	level 2 and 3 blocking in solve
int	pivot_control	pivot control: • 1 Threshold Partial Pivoting is desired • 2 Threshold Rook Pivoting is desired • 3 Threshold Complete Pivoting is desired • 4 Threshold Symmetric Pivoting is desired • 5 Threshold Diagonal Pivoting is desired
int	pivot_search_limit	number of rows/columns pivot selection restricted to (0 = no restriction)
int	minimum_size_for_btf	the minimum permitted size of blocks within the block-triangular form
int	max_iterative_refinements	maximum number of iterative refinements allowed
bool	stop_if_singular	stop if the matrix is found to be structurally singular
real_wp_	array_increase_factor	factor by which arrays sizes are to be increased if they are too small
real_wp_	switch_to_full_code_density	switch to full code when the density exceeds this factor
real_wp_	array_decrease_factor	if previously allocated internal workspace arrays are greater than array_decrease_factor times the currently required sizes, they are reset to current requirements
real_wp_	relative_pivot_tolerance	pivot threshold
real_wp_	absolute_pivot_tolerance	any pivot small than this is considered zero
real_wp_	zero_tolerance	any entry smaller than this in modulus is reset to zero
real_wp_	acceptable_residual_relative	refinement will cease as soon as the residual $\|\|Ax-b\|\|$ falls below max(acceptable_residual_relative $*$ $\|\|b\|\|$, acceptable_residual_absolute)
real_wp_	acceptable_residual_absolute	see acceptable_residual_relative
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'

3.1.1.2 struct uls_inform_type

inform derived type as a C struct

Examples

ulst.c, and ulstf.c.

Data Fields

int	status	reported return status: • 0 success • -1 allocation error • -2 deallocation error • -3 matrix data faulty (m $<$ 1, n $<$ 1, ne $<$ 0) • -26 unknown solver • -29 unavailable option • -31 input order is not a permutation or is faulty in some other way • -32 error with integer workspace • -33 error with real workspace • -50 solver-specific error; see the solver's info parameter
int	alloc_status	STAT value after allocate failure.
char	bad_alloc[81]	name of array which provoked an allocate failure
int	more_info	further information on failure
int	out_of_range	number of indices out-of-range
int	duplicates	number of duplicates
int	entries_dropped	number of entries dropped during the factorization
int	workspace_factors	predicted or actual number of reals and integers to hold factors
int	compresses	number of compresses of data required
int	entries_in_factors	number of entries in factors
int	rank	estimated rank of the matrix
int	structural_rank	structural rank of the matrix
int	pivot_control	pivot control: • 1 Threshold Partial Pivoting has been used • 2 Threshold Rook Pivoting has been used • 3 Threshold Complete Pivoting has been desired • 4 Threshold Symmetric Pivoting has been desired • 5 Threshold Diagonal Pivoting has been desired
int	iterative_refinements	number of iterative refinements performed
bool	alternative	has an "alternative" y: A^T y = 0 and yT b $>$ 0 been found when trying to solve A x = b ?
struct gls_ainfo	gls_ainfo	the output arrays from GLS
struct gls_finfo	gls_finfo	see gls_ainfo
struct gls_sinfo	gls_sinfo	see gls_ainfo
struct ma48_ainfo	ma48_ainfo	the output arrays from MA48
struct ma48_finfo	ma48_finfo	see ma48_ainfo
struct ma48_sinfo	ma48_sinfo	see ma48_ainfo

3.1.2 Function Documentation

3.1.2.1 uls_initialize()

```
void uls_initialize (
            const char solver[],
            void ** data,
            struct uls_control_type * control,
            int * status )
```

Set default control values and initialize private data

Select solver, set default control values and initialize private data

Parameters

in	*solver*	is a one-dimensional array of type char that specifies the solver package that should be used to factorize the matrix A. It should be one of 'gls', 'ma28' or 'ma48; lower or upper case variants are allowed.
in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see uls_control_type)
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. • -26. The requested solver is not available.

Examples

ulst.c, and ulstf.c.

3.1.2.2 uls_read_specfile()

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

Parameters

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

3.1.2.3 uls_factorize_matrix()

```
void uls_factorize_matrix (
            struct uls_control_type * control,
            void ** data,
            int * status,
            int m,
            int n,
            const char type[],
            int ne,
            const real_wp_ val[],
            const int row[],
            const int col[],
            const int ptr[] )
```

Import matrix data into internal storage prior to solution, analyse the sparsity patern, and subsequently factorize the matrix

Parameters

`in`	*control*	is a struct whose members provide control paramters for the remaining prcedures (see uls_control_type)
`in,out`	*data*	holds private internal data
`out`	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 0. The import, analysis and factorization were conducted 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. • -3. The restrictions n $>$ 0 and m$>$ 0 or requirement that the matrix type must contain the relevant string 'dense', 'coordinate' or 'sparse_by_rows has been violated. • -26. The requested solver is not available. • -29. This option is not available with this solver. • -32. More than control.max integer factor size words of internal integer storage are required for in-core factorization. • -50. A solver-specific error occurred; check the solver-specific information component of inform along with the solver's documentation for more details.
`in`	*m*	is a scalar variable of type int, that holds the number of rows in the unsymmetric matrix A.
`in`	*n*	is a scalar variable of type int, that holds the number of columns in the unsymmetric matrix A.

Parameters

in	*type*	is a one-dimensional array of type char that specifies the unsymmetric 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 A in the sparse co-ordinate storage scheme. It need not be set for any of the other schemes.
in	*val*	is a one-dimensional array of size ne and type double, that holds the values of the entries of the matrix A in any of the supported storage schemes.
in	*row*	is a one-dimensional array of size ne and type int, that holds the row indices of the matrix 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.
in	*col*	is a one-dimensional array of size ne and type int, that holds the column indices of the matrix A in either the sparse co-ordinate, or the sparse row-wise storage scheme. It need not be set when the dense storage schemes is used, and in this case can be NULL.
in	*ptr*	is a one-dimensional array of size m+1 and type int, that holds the starting position of each row of the matrix A, as well as the total number of entries plus one, 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

ulst.c, and ulstf.c.

3.1.2.4 uls_reset_control()

```
void uls_reset_control (
             struct uls_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 uls_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.

Examples

ulst.c, and ulstf.c.

3.1.2.5 uls_solve_system()

```
void uls_solve_system (
            void ** data,
            int * status,
            int m,
            int n,
            real_wp_ sol[],
            bool trans )
```

Solve the linear system $Ax = b$ or $A^T x = b$.

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. • -34. The package PARDISO failed; check the solver-specific information components inform.pardiso iparm and inform.pardiso_dparm along with PARDISO's documentation for more details. • -35. The package WSMP failed; check the solver-specific information components inform.wsmp_iparm and inform.wsmp dparm along with WSMP's documentation for more details.
in	*m*	is a scalar variable of type int, that holds the number of rows in the unsymmetric matrix A.
in	*n*	is a scalar variable of type int, that holds the number of columns in the unsymmetric matrix A.
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.
in	*trans*	is a scalar variable of type bool, that specifies whether to solve the equation $A^T x = b$ (trans=true) or $Ax = b$ (trans=false).

Examples

ulst.c, and ulstf.c.

3.1.2.6 uls_information()

```
void uls_information (
            void ** data,
            struct uls_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 uls_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

ulst.c, and ulstf.c.

3.1.2.7 uls_terminate()

```
void uls_terminate (
            void ** data,
            struct uls_control_type * control,
            struct uls_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 uls_control_type)
out	*inform*	is a struct containing output information (see uls_inform_type)

Examples

ulst.c, and ulstf.c.

Chapter 4

Example Documentation

4.1 ulst.c

This is an example of how to use the package in conjunction with the sparse linear solver `sils`. A variety of supported matrix storage formats are illustrated.

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

```
/* ulst.c */
/* Full test for the ULS C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include "galahad_uls.h"
int maxabsarray(double a[],int n, double *maxabs);
int main(void) {
    // Derived types
    void *data;
    struct uls_control_type control;
    struct uls_inform_type inform;
    // Set problem data
    int m = 5; // column dimension of A
    int n = 5; // column dimension of A
    int ne = 7; // number of entries of A
    int dense_ne = 25; // number of elements of A as a dense matrix
    int row[] = {0, 1, 1, 2, 2, 3, 4}; // row indices
    int col[] = {0, 0, 4, 1, 2, 2, 3}; // column indices
    int ptr[] = {0, 1, 3, 5, 6, 7}; // pointers to indices
    double val[] = {2.0, 3.0, 6.0, 4.0, 1.0, 5.0, 1.0}; // values
    double dense[] = {2.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 6.0,
                      0.0, 4.0, 1.0, 0.0, 0.0, 0.0, 0.0, 5.0, 0.0, 0.0,
                      0.0, 0.0, 0.0, 1.0, 0.0};
    double rhs[] = {2.0, 33.0, 11.0, 15.0, 4.0};
    double rhst[] = {8.0, 12.0, 23.0, 5.0, 12.0};
    double sol[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int i, status;
    double x[n];
    double error[n];
    _Bool trans;
    double norm_residual;
    double good_x = pow( DBL_EPSILON, 0.3333 );
    printf(" C sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    printf(" storage          RHS   refine   RHST  refine\n");
    for( int d=1; d <= 3; d++){
        // Initialize ULS - use the gls solver
        uls_initialize( "gls", &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = false; // Fortran sparse matrix indexing
        switch(d){ // import matrix data and factorize
            case 1: // sparse co-ordinate storage
                printf(" coordinate     ");
                uls_factorize_matrix( &control, &data, &status, m, n,
                                    "coordinate", ne, val, row, col, NULL );
                break;
            case 2: // sparse by rows
                printf(" sparse by rows ");
                uls_factorize_matrix( &control, &data, &status, m, n,
                                      "sparse_by_rows", ne, val, NULL, col, ptr );
                break;
            case 3: // dense
                printf(" dense          ");
                uls_factorize_matrix( &control, &data, &status, m, n, "dense",
                                        dense_ne, dense, NULL, NULL, NULL );
                break;
            }
        // Set right-hand side and solve the system A x = b
        for(i=0; i<n; i++) x[i] = rhs[i];
        trans = false;
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("   ok  ");
          }else{
            printf("  fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // printf("sol: ");
        // for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // resolve, this time using iterative refinement
        control.max_iterative_refinements = 1;
        uls_reset_control( &control, &data, &status );
        for(i=0; i<n; i++) x[i] = rhs[i];
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("    ok  ");
          }else{
            printf("   fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // Set right-hand side and solve the system A^T x = b
        for(i=0; i<n; i++) x[i] = rhst[i];
        trans = true;
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("   ok  ");
          }else{
            printf("  fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // printf("sol: ");
        // for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // resolve, this time using iterative refinement
        control.max_iterative_refinements = 1;
        uls_reset_control( &control, &data, &status );
        for(i=0; i<n; i++) x[i] = rhst[i];
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("    ok  ");
          }else{
            printf("   fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // Delete internal workspace
        uls_terminate( &data, &control, &inform );
        printf("\n");
    }
}
```

```
int maxabsarray(double a[],int n, double *maxabs)
 {
    int i;
    double b,max;
    max=abs(a[0]);
    for(i=1; i<n; i++)
    {
        b = abs(a[i]);
    if(max<b)
          max=b;
    }
    *maxabs=max;
 }
```

4.2 ulstf.c

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

```
/* ulstf.c */
/* Full test for the ULS C interface using Fortran sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include "galahad_uls.h"
int maxabsarray(double a[],int n, double *maxabs);
int main(void) {
    // Derived types
    void *data;
    struct uls_control_type control;
    struct uls_inform_type inform;
    // Set problem data
    int m = 5; // column dimension of A
    int n = 5; // column dimension of A
    int ne = 7; // number of entries of A
    int dense_ne = 25; // number of elements of A as a dense matrix
    int row[] = {1, 2, 2, 3, 3, 4, 5}; // row indices
    int col[] = {1, 1, 5, 2, 3, 3, 4}; // column indices
    int ptr[] = {1, 2, 4, 6, 7, 8}; // pointers to indices
    double val[] = {2.0, 3.0, 6.0, 4.0, 1.0, 5.0, 1.0}; // values
    double dense[] = {2.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 6.0,
                      0.0, 4.0, 1.0, 0.0, 0.0, 0.0, 0.0, 5.0, 0.0, 0.0,
                      0.0, 0.0, 0.0, 1.0, 0.0};
    double rhs[] = {2.0, 33.0, 11.0, 15.0, 4.0};
    double rhst[] = {8.0, 12.0, 23.0, 5.0, 12.0};
    double sol[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int i, status;
    double x[n];
    double error[n];
    _Bool trans;
    double norm_residual;
    double good_x = pow( DBL_EPSILON, 0.3333 );
    printf(" Fortran sparse matrix indexing\n\n");
    printf(" basic tests of storage formats\n\n");
    printf(" storage          RHS    refine   RHST  refine\n");
    for( int d=1; d <= 3; d++){
        // Initialize ULS - use the gls solver
        uls_initialize( "gls", &data, &control, &status );
        // Set user-defined control options
        control.f_indexing = true; // Fortran sparse matrix indexing
        switch(d){ // import matrix data and factorize
            case 1: // sparse co-ordinate storage
                printf(" coordinate     ");
                uls_factorize_matrix( &control, &data, &status, m, n,
                                    "coordinate", ne, val, row, col, NULL );
                break;
            case 2: // sparse by rows
                printf(" sparse by rows ");
                uls_factorize_matrix( &control, &data, &status, m, n,
                                    "sparse_by_rows", ne, val, NULL, col, ptr );
                break;
            case 3: // dense
                printf(" dense          ");
                uls_factorize_matrix( &control, &data, &status, m, n,
                                     "dense", dense_ne, dense, NULL, NULL, NULL );
                break;
            }
        // Set right-hand side and solve the system A x = b
        for(i=0; i<n; i++) x[i] = rhs[i];
        trans = false;
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("   ok  ");
          }else{
            printf("  fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // printf("sol: ");
        // for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // resolve, this time using iterative refinement
        control.max_iterative_refinements = 1;
        uls_reset_control( &control, &data, &status );
        for(i=0; i<n; i++) x[i] = rhs[i];
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("    ok  ");
          }else{
            printf("   fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // Set right-hand side and solve the system A^T x = b
        for(i=0; i<n; i++) x[i] = rhst[i];
        trans = true;
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("   ok  ");
          }else{
            printf("  fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // printf("sol: ");
        // for( int i = 0; i < n; i++) printf("%f ", x[i]);
        // resolve, this time using iterative refinement
        control.max_iterative_refinements = 1;
        uls_reset_control( &control, &data, &status );
        for(i=0; i<n; i++) x[i] = rhst[i];
        uls_solve_system( &data, &status, m, n, x, trans );
        uls_information( &data, &inform, &status );
        if(inform.status == 0){
          for(i=0; i<n; i++) error[i] = x[i]-sol[i];
          status = maxabsarray( error, n, &norm_residual );
          if(norm_residual < good_x){
            printf("    ok  ");
          }else{
            printf("   fail ");
          }
        }else{
            printf(" ULS_solve exit status = %1i\n", inform.status);
        }
        // Delete internal workspace
        uls_terminate( &data, &control, &inform );
        printf("\n");
    }
}
int maxabsarray(double a[],int n, double *maxabs)
 {
    int i;
    double b,max;
    max=abs(a[0]);
    for(i=1; i<n; i++)
    {
        b = abs(a[i]);
    if(max<b)
          max=b;
    }
    *maxabs=max;
 }
```

Index