1 SUMMARY

HSL_MC69 offers **routines for converting matrices held in a number of sparse matrix formats to the compressed sparse column (CSC) format** used by many HSL routines. This format requires the entries within each column of A to be **ordered by increasing row index**. For symmetric, skew symmetric or Hermitian matrices, only entries in the **lower triangle** are held. This format is the one used by many of the recent HSL packages; we shall refer to it as the **standard HSL format**.

Routines are offered for **converting** matrices held in lower or upper compressed sparse column format or in lower or upper compressed sparse row format or in coordinate format, and for **verification** and **correction** of matrices believed to already be in standard HSL format. The conversion routines check the user-supplied data for errors and handle duplicate entries (they are summed) and out-of-range entries (they are discarded).

ATTRIBUTES — Version: 1.4.2 (28 July 2022). **Types:** Real (single, double), Complex (single, double). **Calls:** None **Original date:** January 2011. **Origin:** J.D. Hogg, Rutherford Appleton Laboratory. **Language:** Fortran 2003 subset (F95 + TR15581 + C interoperability). **Interfaces:** Fortran, C.

2 HOW TO USE THE PACKAGE

2.1 C interface to Fortran code

This package is written in Fortran and a wrapper is provided for C programmers. This wrapper only implements a (large) subset of the full functionality available via the Fortran interface.

The wrapper will automatically convert between 0-based (C) and 1-based (Fortran) indexing (see Section 2.4), 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. The conversion may be disabled by setting the argument `findex` to a non-zero value and supplying all data using 1-based indexing. Except where stated, this document describes all arrays in 0-based (C) indexing only.

The wrapper uses the Fortran 2003 interoperability features. **Matching C and Fortran compilers must be used**, for example gcc and gfortran, or icc and ifort. If the Fortran compiler is not used to link the user's program, then additional Fortran compiler libraries may need to be linked explicitly.

2.2 Routines available

A verification routine for matrices in standard HSL format can be found on page 4. Routines for handling user-supplied matrices that are held in other formats may be found as specified below. **The section on each format is designed to be self contained**, thus users only need to read the section relevant to them.

2.3 The header file

Access to the package requires inclusion of a header file

Single precision version
```
#include "hsl_mc69s.h"
```
Double precision version
```
#include "hsl_mc69d.h"
```
Complex version
```
#include "hsl_mc69c.h"
```
Double complex version
```
#include "hsl_mc69z.h"
```

If more than one version is to be used within the same code, one of the postfixes _s, _d, _c, or _z must be added to the end of all subroutine calls. For example:

```
mc69_print_s(...) /* Single precision call to mc69_print() */
mc69_print_d(...) /* Double precision call to mc69_print() */
mc69_print_c(...) /* Complex call to mc69_print()          */
mc69_print_z(...) /* Double Complex call to mc69_print()   */
```

2.3.1 Package types

The complex versions require C99 support for the `double complex` and `float complex` types. The real versions do not require C99 support.

We use the following type definitions in the different versions of the package:

Single precision version

```
typedef float pkgtype
```

Double precision version

```
typedef double pkgtype
```

Complex version

```
typedef float complex pkgtype
```

Double complex version

```
typedef double complex pkgtype
```

2.4 Fortran indexing

If `findex` is non-zero, Fortran indexing is used to store the matrix. This means that any integer values stored in an array that is passed to a function must be 1 greater than for C indexing. However, C indexing must still be used to access the array.

Consider the following matrix:

$$\begin{pmatrix} 1 & & 3 \\ & 2 & 4 \end{pmatrix}$$

It may be stored in CSC format as follows:

findex=0 (C)		**findex=1 (Fortran)**	
ptr[0] = 0		ptr[0] = 1	
ptr[1] = 1		ptr[1] = 2	
ptr[2] = 2		ptr[2] = 3	
ptr[3] = 4		ptr[3] = 5	
row[0] = 0	val[0] = 1	row[0] = 1	val[0] = 1
row[1] = 1	val[1] = 2	row[1] = 2	val[1] = 2
row[2] = 0	val[2] = 3	row[2] = 1	val[2] = 3
row[3] = 1	val[3] = 4	row[3] = 2	val[3] = 4

2.5 Matrix type constants

The following enumeration is defined in the header file:

```
typedef enum hsl_matrix_type {
    /* Undefined or Unknown matrix */
    HSL_MATRIX_UNDEFINED          = 0,

    /* Real matrices */
    HSL_MATRIX_REAL_RECT          = 1,   /* real rectangular */
    HSL_MATRIX_REAL_UNSYM         = 2,   /* real unsymmetric */
    HSL_MATRIX_REAL_SYM_PSDEF     = 3,   /* real symmetric, positive definite */
    HSL_MATRIX_REAL_SYM_INDEF     = 4,   /* real symmetric, indefinite */
    HSL_MATRIX_REAL_SKEW          = 6,   /* real skew symmetric */

    /* Complex matrices */
    HSL_MATRIX_CPLX_RECT          = -1,  /* complex rectangular */
    HSL_MATRIX_CPLX_UNSYM         = -2,  /* complex unsymmetric */
    HSL_MATRIX_CPLX_HERM_PSDEF    = -3,  /* complex Hermitian, positive definite */
    HSL_MATRIX_CPLX_HERM_INDEF    = -4,  /* complex Hermitian, indefinite */
    HSL_MATRIX_CPLX_SYM           = -5,  /* complex symmetric */
    HSL_MATRIX_CPLX_SKEW          = -6   /* complex skew symmetric */
} hsl_matrix_type;
```

2.6 Matrices held in standard HSL format

The following routines handle a matrix A held in standard HSL format (that is, CSC format with the entries within each column ordered by increasing row index). For symmetric, skew symmetric or Hermitian matrices, only the **lower triangle** is held. There is no requirement that zero entries on the diagonal be explicitly included.

A valid matrix of this form has no out-of-range or duplicate entries, and is stored as a series of compressed columns using the following data:

`m` is an `int` scalar that holds the number of rows in A.

`n` is an `int` scalar that holds the number of columns in A.

`ptr` is a rank-one `int` array. The first `n` values must be set such that `ptr[j]` holds the position in `row` of the first entry in column `j` and `ptr[n]` must be the total number of entries.

`row` is a rank-one `int` array. The first `ptr[n]` entries hold the row indices of the entries of A, with the row indices for the entries in first column preceding those for the second, and so on. The indices within each column **must** be in increasing order.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one `pkgtype` array. `val[k]` must hold the value of the entry in `row[k]`.

2.6.1 To verify a matrix is in standard HSL format

To verify a matrix is in standard HSL format, or to identify why it is not, the user may make the following call. Note that this routine does **not** handle duplicates or out-of-range entries (they are flagged as errors). It is intended for debugging rather than for use in a performance code.

```
int mc69_verify(const int unit, const hsl_matrix_type type, const int findex,
   const int m, const int n, const int ptr[], const int row[], const pkgtype val[],
   int *more);
```

Arguments:

`unit` holds the Fortran unit for output. If `unit` $\geq$ `0`, error messages are printed on unit `unit`, otherwise they are suppressed.

`type` describes the type of the matrix. It must have one of the values given in Section 2.5. If it has value `0` (`HSL_MATRIX_UNDEFINED`) requirements that depend on the matrix type will not be checked. For positive-definite matrices, the positive-definite property is not tested (except that diagonal entries must be present and positive).

`findex` specifies the findex array index. If the arrays `ptr` and `row` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr` and `row` is taken internally by the function.

`m`, `n`, `ptr` and `row` must be set by the user to hold A in standard HSL format as described in Section 2.6.

`val` may be NULL. If it is not NULL, `val[k]` must hold the value of the entry in `row[k]`.

`more` is used to provide further information if the matrix is not in HSL standard format; see list of error returns from this function.

Return value:
A value of 0 indicates the matrix is in standard HSL format. Negative values are associated with an error, and will take one of the following values:

-1 Allocation error. more will be set to the Fortran stat value from the failed allocate.

-2 Invalid value of type.

-3 m<0 or n<0.

-4 |type| > 1 (square matrix) but m≠n.

-5 ptr[0]<0. more is set to ptr[0].

-6 ptr is not monotonically increasing. more is set to the least value of i such that ptr[i] < ptr[i-1].

-7 Entries within one or more columns are not sorted by increasing row index. more is set to the first index such that row[more] < row[more-1] and both are in the same column.

-8 row contains one or more out-of-range entries. more holds the index of the first out-of-range entry in row.

-9 row contains one or more duplicate entries. more is set such that row[more] and row[more+1] are the first pair of duplicate entries.

-11 |type| = 3 (positive-definite case) but one or more diagonal entries are missing or are not positive. more is set to the index of the first column with such a diagonal entry.

-12 type = −3 or -4 (Hermitian case) but one or more diagonal entries have non-zero imaginary part. more is set to the index of the first column with such a diagonal entry.

-14 matrix is symmetric, skew-symmetric or Hermitian and an entry is present in the upper triangle or on the diagonal of a skew-symmetric matrix. more is set so that row[more] is the first such entry.

2.6.2 To print a matrix in standard HSL format

To print a matrix in standard HSL format (or print a summary of one), the user may call mc69_print. Note that output will use Fortran numbering.

```
void mc69_print(const int unit, const int lines, const hsl_matrix_type type,
   const int findex, const int m, const int n, const int ptr[], const int row[],
   const pkgtype val[])
```

Arguments:
unit holds the Fortran unit for output. If unit ≥ 0, error messages are printed on unit unit, otherwise they are suppressed.

lines controls the number of lines of output used. If lines ≥ 0, a summary of the matrix will be printed of not more than lines lines. Otherwise, the whole matrix will be printed.

type describes the type of the matrix. It must have one of the values given in Section 2.5.

findex specifies the findex array index. If the arrays ptr and row start numbering at 0 in the C style, findex must be set to 0. If the arrays instead start numbering at 1 in the Fortran style, findex must be non-zero. If findex=0, an extra copy of ptr and row is taken internally by the function.

m, n, ptr and row must be set by the user to hold A in standard HSL format as described in Section 2.6.

val may be NULL. If it is not NULL, it must be of size ptr[n] and val[k] must hold the value of the entry in row[k].

2.6.3 Example

Usage of the routines in this section will be demonstrated using the following matrix

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0 \end{pmatrix}.$$

The following code reads a matrix in HSL standard form, verifies the matrix is a valid using `mc69_verify`, and then displays the matrix using `mc69_print`. If provided with the following input (matching the matrix A above) on stdin, the code produces the following output.

2.7 Compressed sparse column format

The following routines handle a matrix stored in compressed sparse column format, with entries only in the lower triangle for symmetric, skew-symmetric or Hermitian matrices. Entries within each column of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included. For a skew-symmetric matrix, no diagonal entries are held.

The input matrix is stored as a series of compressed columns using the following data:

`m` is a scalar of type `int` that holds the number of rows of A.

`n` is a scalar of type `int` that holds the number of columns of A.

`ptr` is a rank-one array of type `int`. The first `n` values must be set such that `ptr[j]` holds the position in `row` of the first entry in column `j` and `ptr[n]` must be the total number of entries.

`row` is a rank-one array of type `INTEGER`. The first `ptr[n]` entries hold the row indices of the entries, with the row indices for the entries in first column preceding those for the second, and so on. The indices within each column may be unordered.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `row[k]`.

2.7.1 To perform an *in-place* conversion from compressed sparse column format to standard HSL format

To convert a matrix held in compressed sparse column format to standard HSL format **in-place** (that is, the user's data is overwritten), the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are removed). For symmetric, skew-symmetric and Hermitian matrices, entries in the upper triangle are removed. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range entries, and are removed.

```
int mc69_cscl_clean(const int unit, const hsl_matrix_type type, const int findex,
   const int m, const int n, int ptr[], int row[], pkgtype val[],
   int *noor, int *ndup, int *lmap, int *map[])
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`findex` specifies the findex array index. If the arrays `ptr` and `row` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr` and `row` is taken internally by the function.

`m`, `n`, `ptr` and `row` must be set by the user to hold A in compressed sparse column format, as described in Section 2.7. On exit, `ptr` and `row` will have been modified to hold the matrix in standard HSL format.

`val` may be NULL. If not NULL, on input the first `ptr[n]` entries must be set so that `val[k]` holds the value of the entry in `row[k]`. On exit, it will hold the (potentially modified) values of the matrix entries corresponding to those of the array `row`.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_cscl_clean, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
On exit, a value of 0 indicates successful conversion. Positive values indicate successful conversion but a warning was issued. Negative values are associated with an error; see Section 2.7.4 for details.

2.7.2 To perform an *out-of-place* conversion from compressed sparse column format to standard HSL format

To convert a matrix held in lower compressed sparse column format to standard HSL format, the user may make a call of the following form. This routine leaves the user's data unchanged. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). For symmetric, skew-symmetric and Hermitian matrices, entries in the upper triangle are discarded. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range entries, and are discarded.

```
int mc69_cscl_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int m, const int n, const int ptr_in[],
   const int row_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[]);
```

Arguments:
unit holds the Fortran unit number for output. If unit $\geq$ 0, error and warning messages are written to unit unit, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if findex$\neq$0), regardless of the value of findex.

type describes the type of matrix. It must take one of the values described in Section 2.5. If this argument has value 0 (HSL_MATRIX_UNDEFINED), the matrix will be treated as if it were rectangular.

findex specifies the findex array index. If the arrays ptr, row, ptr_out and row_out start numbering at 0 in the C style, findex must be set to 0. If the arrays instead start numbering at 1 in the Fortran style, findex must be non-zero. If findex=0, an extra copy of ptr, row, ptr_out and row_out is taken internally by the function.

m, n, ptr_in and row_in must be set by the user to hold A in compressed sparse column format, as described in Section 2.7.

val_in may be NULL. If it is not NULL, on input the first ptr_in[n] entries must be set so that val_in[k] holds the value of the entry col_in[k]. If val_in is NULL, val_out must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out is an optional INTENT(OUT) rank-one allocatable array of package type. If not NULL, on exit it will be allocated and the first ptr_out[n] entries will be set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of *A* following the call to mc69_cscl_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
On exit, a value of 0 indicates successful conversion. Positive values indicate successful conversion but a warning was issued. Negative values are associated with an error; see Section 2.7.4 for details.

2.7.3 To set values of *A* following a conversion

The user may want to change the values of the entries of *A* following a successful call to mc69_cscl_clean or mc69_cscl_convert. Alternatively, the user may want to include matrix values after a call to mc69_cscl_clean or mc69_cscl_convert in which matrix values were not passed. This can be done by making a call of the following form, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
   const pkgtype val_in[], const int ne, pkgtype val_out[])}
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated map.

lmap must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated map.

map must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated it.

val_in must have size at least the value of ptr[n] on the call to mc69_cscl_clean (or ptr_in[n] for a call to mc69_cscl_convert). It must be set by the user to hold the new values of the entries of *A* matching the original matrix that was input to mc69_cscl_clean or mc69_cscl_convert.

ne must be set to the number of entries in the HSL standard form matrix. If using C indexing, this is ptr[n] on exit from mc69_cscl_clean or ptr_out[n] on exit from mc69_cscl_convert (if using Fortran based indexing, subtract 1 from these values).

val_out must have size at least ne. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.6.

2.7.4 Return codes

A successful return from a call to mc69_cscl_clean or mc69_cscl_convert is indicated by flag taking the value 0. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 m<0 or n<0.

-4 |type| > 1 (square matrix) but m≠n.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a column are out of range.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| ≠ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| ≠ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.7.5 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in Compressed Sparse Column form, and then performs an *in-place* conversion to HSL standard form using mc69_cscl_clean. If provided with the following input (matching the matrix A above) on stdin, the code produces the following output.

For an *out-of-place* conversion, the following code calling mc69_cscl_convert may be used instead. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

2.8 Symmetric, skew symmetric and Hermitian matrices in upper compressed sparse column format

The following routines handle a symmetric, skew-symmetric or Hermitian matrix stored in upper compressed sparse column format (only entries in the upper triangle are stored). Entries within each column of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included.
The input matrix is stored as a series of compressed columns using the following data:

`n` is a scalar of type `int` that holds the order of A.

`ptr` is a rank-one array of type `int`. The first `n` values must be set such that `ptr[j]` holds the position in `row` of the first entry in column `j` and `ptr[n]` must be the total number of entries.

`row` is a rank-one array of type `int`. The first `ptr[n]` entries hold the row indices of the entries in the **upper triangle** of A, with the row indices for the entries in the first column preceding those in the second, and so on. The indices within each column may be unordered.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `row[k]`.

2.8.1 To perform a conversion from upper compressed sparse column format to standard HSL format

To convert a symmetric, skew-symmetric or Hermitian matrix held in upper compressed sparse column format to standard HSL format, the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). Entries in the lower triangle are discarded. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range entries, and are discarded.

```
int mc69_cscu_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int n, const int ptr_in[],
   const int row_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[])
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5 for a symmetric, skew-symmetric or Hermitian matrix.

`findex` specifies the findex array index. If the arrays `ptr_in`, `row_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr_in`, `row_in`, `ptr_out` and `row_out` is taken internally by the function.

`n`, `ptr_in` and `row_in` must be set by the user to hold A in upper compressed sparse row format, as described in Section 2.8.

`val_in` may be NULL. If not NULL, on input the first `ptr_in[n]` entries must be set so that `val_in[k]` holds the value of the entry `row_in[k]`. If `val_in` is NULL, `val_out` must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first ptr_out[n] entries will be set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_cscu_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return is indicated by a value of 0. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 n<0.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a column are out of range.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| $\neq$ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| $\neq$ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.8.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to mc69_cscu_convert. Alternatively, the user may want to include matrix values after a call to mc69_cscu_convert with matrix values not not NULL. This can be done by making a call to mc69_set_values, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_cscu_convert that generated map.

lmap must be unchanged since the call to mc69_cscu_convert that generated map.

map must be unchanged since the call to mc69_cscu_convert that generated it.

val_in must have size at least the value of ptr_in[n] on the call to mc69_cscu_convert. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to mc69_cscu_convert.

ne must be set to the number of entries in the output matrix from the call to mc69_cscu_convert. If using C indexing, this is the value of ptr_out[n] on exit from mc69_cscu_convert (for Fortran indexing, subtract 1).

val_out must have size at least the value of ptr_out[n] on exit from mc69_cscu_convert. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.8.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in upper Compressed Sparse Column form, and then converts it to HSL standard format using mc69_cscu_convert. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

2.9 Symmetric, skew symmetric and Hermitian matrices in full compressed sparse column format

The following routines handle a symmetric, skew symmetric or Hermitian matrix stored in full compressed sparse column format (entries in both the lower and upper triangles are supplied by the user). Entries within each column of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included.

The input matrix is stored as a series of compressed columns using the following data:

`n` is a scalar of type `INTEGER` that holds the order of A.

`ptr` is a rank-one array of type `INTEGER`. The first `n` values must be set such that `ptr[j]` holds the position in `row` of the first entry in column `j` and `ptr[n]` must be the total number of entries.

`row` is a rank-one array of type `INTEGER`. The first `ptr[n]` entries hold the row indices of the entries of A, with the row indices for the entries in column 1 preceding those for column 2, and so on. If a non-diagonal entry (i, j) is present, its counterpart (j, i) must also be present. The indices within each column may be unordered.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `row[k]`.

2.9.1 To convert from full compressed sparse column format to standard HSL format

To convert a symmetric, skew-symmetric or Hermitian matrix held in full compressed sparse column format to standard HSL format the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). Entries in the lower triangle are ignored, except to check that there are the same number of entries in both the lower and upper triangles. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range and are discarded.

```
int mc69_csclu_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int n, const int ptr_in[],
   const int row_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[]);
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5 for a symmetric, skew-symmetric or Hermitian matrix.

`findex` specifies the findex array index. If the arrays `ptr_in`, `row_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr_in`, `row_in`, `ptr_out` and `row_out` is taken internally by the function.

`n`, `ptr_in` and `row_in` must be set by the user to hold A in full compressed sparse column format, as described in Section 2.9.

`val_in` may be NULL. If not NULL, the first `ptr_in[n]` entries must be set so that `val_in[k]` holds the value of the entry `row_in[k]`. If `val_in` is NULL, `val_out` must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first ptr_out[n] entries are set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_csclu_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return indicated by value of 0 or above, with non-zero values indicating a warning. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 n<0.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a column are out of range.

-13 Number of in-range entries in lower and upper triangles do not match.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| $\neq$ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| $\neq$ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.9.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to mc69_csclu_convert. Alternatively, the user may want to include matrix values after a call to mc69_csclu_convert with matrix values not not NULL. This can be done by making a call to mc69_set_values, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_csclu_convert that generated map.

lmap must be unchanged since the call to mc69_csclu_convert that generated map.

map must be unchanged since the call to mc69_csclu_convert that generated it.

val_in must have size at least the value of ptr_in[n] on the call to mc69_csclu_convert. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to mc69_csclu_convert.

ne must be set to the number of entries in the output matrix from the call to mc69_csclu_convert. If using C indexing, this is the value of ptr_out[n] on exit from mc69_csclu_convert (for Fortran indexing, subtract 1).

val_out must have size at least the value of ptr_out[n] on exit from mc69_csclu_convert. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.9.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in full Compressed Sparse Column form, and then converts it to HSL standard format using mc69_csclu_convert. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

2.10 Matrices in compressed sparse row format

The following routines handle a matrix stored in compressed sparse row format, with entries only in the lower triangle for symmetric, skew-symmetric or Hermitian matrices. Entries within each row of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included.

The input matrix is stored as a series of compressed rows using the following data:

`m` is a scalar of type `INTEGER` that holds the number of rows of A.

`n` is a scalar of type `INTEGER` that holds the number of columns of A.

`ptr` is a rank-one array of type `INTEGER`. The first `m` values must be set such that `ptr[j]` holds the position in `col` of the first entry in row `j` and `ptr[m]` must be the total number of entries.

`col` is a rank-one array of type `INTEGER`. The first `ptr[m]` entries hold the column indices of the entries in A, with the column indices for the entries in row 0 preceding those for row 1, and so on. For symmetric, skew symmetric and Hermitian matrices only entries in the lower triangle should be stored. The indices within each row may be unordered.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `col[k]`.

2.10.1 To perform a conversion from compressed sparse row format to standard HSL format

To convert a matrix held in compressed sparse row format to standard HSL format, the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). For symmetric, skew-symmetric and Hermitian matrices, entries in the upper triangle are discarded as out-of-range. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range entries, and are discarded.

```
int mc69_csrl_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int m, const int n, const int ptr_in[],
   const int col_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[]);
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`findex` specifies the findex array index. If the arrays `ptr_in`, `col_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr_in`, `col_in`, `ptr_out` and `row_out` is taken internally by the function.

`m`, `n`, `ptr_in` and `col_in` must be set by the user to hold A in compressed sparse row format, as described in Section 2.10.

val_in may be NULL. If not NULL, on input the first ptr_in[m] entries must be set so that val_in[k] holds the value of the entry col_in[k]. If val_in is NULL, val_out must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first ptr_out[n] entries will be set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_csrl_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return indicated by value of 0 or above, with non-zero values indicating a warning. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 m$<$0 or n$<$0.

-4 |type| $>$ 1 (square matrix) but m$\neq$n.

-5 ptr[0]$<$0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a row are out of range.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| $\neq$ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| $\neq$ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.10.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to `mc69_csrl_convert`. Alternatively, the user may want to include matrix values after a call to `mc69_csrl_convert` with matrix values not not NULL. This can be done by making a call to `mc69_set_values`, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

`type` describes the type of matrix. It must be unchanged since the call to `mc69_csrl_convert` that generated `map`.

`lmap` must be unchanged since the call to `mc69_csrl_convert` that generated `map`.

`map` must be unchanged since the call to `mc69_csrl_convert` that generated it.

`val_in` must have size at least the value of `ptr_in[m]` on the call to `mc69_csrl_convert`. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to `mc69_csrl_convert`.

`ne` must be set to the number of entries in the output matrix from the call to `mc69_csrl_convert`. If using C indexing, this is the value of `ptr_out[n]` on exit from `mc69_csrl_convert` (for Fortran indexing, subtract 1).

`val_out` must have size at least the value of `ptr_out[n]` on exit from `mc69_csrl_convert`. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.10.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \quad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in Compressed Sparse Row form, and then converts it to HSL standard format using `mc69_csrl_convert`. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using `mc69_set_values`. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

2.11 Symmetric, skew symmetric and Hermitian matrices in upper compressed sparse row format

The following routines handle symmetric, skew-symmetric of Hermitian matrices stored in upper compressed sparse row format (with entries only in the upper triangle). Entries within each row of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included.
The input matrix is stored as a series of compressed rows using the following data:

`n` is a scalar of type `int` that holds the order of A.

`ptr` is a rank-one array of type `int`. The first `n` values must be set such that `ptr[j]` holds the position in `col` of the first entry in row `j` and `ptr[n]` must be the total number of entries.

`col` is a rank-one array of type `int`. The first `ptr[n]` entries hold the column indices of the entries in A, with the column indices for the entries in row 0 preceding those for row 1, and so on. For symmetric, skew symmetric and Hermitian matrices only entries in the upper triangle should be stored. The indices within each row may be unordered.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `col[k]`.

2.11.1 To perform a conversion from upper compressed sparse row format to standard HSL format

To convert a matrix held in upper compressed sparse row format to standard HSL format, the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). Entries in the lower triangle are discarded. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range entries, and are discarded.

```
int mc69_csru_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int n, const int ptr_in[],
   const int col_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[])
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5 for symmetric, skew-symmetric or Hermitian matrices.

`findex` specifies the findex array index. If the arrays `ptr_in`, `col_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr_in`, `col_in`, `ptr_out` and `row_out` is taken internally by the function.

`n`, `ptr_in` and `col_in` must be set by the user to hold A in upper compressed sparse row format, as described in Section 2.11.

`val_in` may be NULL. If not NULL, on input the first `ptr_in[n]` entries must be set so that `val_in[k]` holds the value of the entry `col_in[k]`. If `val_in` is NULL, `val_out` must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first ptr_out[n] entries will be set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_csru_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return indicated by value of 0 or above, with non-zero values indicating a warning. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 n<0.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a row are out of range.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| $\neq$ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| $\neq$ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.11.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to mc69_csru_convert. Alternatively, the user may want to include matrix values after a call to mc69_csru_convert with matrix values not not NULL. This can be done by making a call to mc69_set_values, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_csru_convert that generated map.

lmap must be unchanged since the call to mc69_csru_convert that generated map.

map must be unchanged since the call to mc69_csru_convert that generated it.

val_in must have size at least the value of ptr_in[n] on the call to mc69_csru_convert. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to mc69_csru_convert.

ne must be set to the number of entries in the output matrix from the call to mc69_csru_convert. If using C indexing, this is the value of ptr_out[n] on exit from mc69_csru_convert (for Fortran indexing, subtract 1).

val_out must have size at least the value of ptr_out[n] on exit from mc69_csru_convert. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.11.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in upper Compressed Sparse Row form, and then converts it to HSL standard format using mc69_csru_convert. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

2.12 Symmetric, skew symmetric and Hermitian matrices in full compressed sparse row format

The following routines handle a symmetric, skew symmetric or Hermitian matrix stored in full compressed sparse row format (entries in both the lower and upper triangles are supplied by the user). Entries within each row of the user-supplied matrix do **not** need to be ordered. There is no requirement that zero entries on the diagonal be explicitly included.

The input matrix is stored as a series of compressed rows using the following data:

`n` is a scalar of type `int` that holds the order of A.

`ptr` is a rank-one array of type `int`. The first `n` values must be set such that `ptr[j]` holds the position in `col` of the first entry in row `j` and `ptr[n]` must be the total number of entries.

`col` is a rank-one array of type `int`. The first `ptr[n]` entries hold the column indices of the entries of A, with the column indices for the entries in row 1 preceding those for row 2, and so on. If a non-diagonal entry (i, j) is present, its counterpart (j, i) must also be present. The indices within each row may be unordered,

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. `val[k]` must hold the value of the entry in `col[k]`.

2.12.1 To convert from full compressed sparse row format to standard HSL format

To convert a symmetric, skew-symmetric or Hermitian matrix held in full compressed sparse row format to standard HSL format the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). Entries in the upper triangle are ignored, except to check that there are the same number of entries in both the lower and upper triangles. For skew-symmetric matrices only, entries on the diagonal are treated as out-of-range and are discarded.

```
int mc69_csrlu_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int n, const int ptr_in[],
   const int col_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[])
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5 for symmetric, skew-symmetric or Hermitian matrices.

`findex` specifies the findex array index. If the arrays `ptr_in`, `col_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `ptr_in`, `col_in`, `ptr_out` and `row_out` is taken internally by the function.

`n`, `ptr_in` and `col_in` must be set by the user to hold A in full compressed sparse column format, as described in Section 2.12.

`val_in` may be NULL. If not NULL, the first `ptr_in[n]` entries must be set so that `val_in[k]` holds the value of the entry `row_in[k]`. If `val_in` is NULL, `val_out` must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first On exit, it is allocated to have size equal to that of row_out and val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_csrlu_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return indicated by value of 0 or above, with non-zero values indicating a warning. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 n<0.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries for a row are out of range.

-13 Number of in-range entries in lower and upper triangles do not match.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| $\neq$ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| $\neq$ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.12.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to mc69_csrlu_convert. Alternatively, the user may want to include matrix values after a call to mc69_csrlu_convert with matrix values not not NULL. This can be done by making a call to mc69_set_values, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_csrlu_convert that generated map.

lmap must be unchanged since the call to mc69_csrlu_convert that generated map.

map must be unchanged since the call to mc69_csrlu_convert that generated it.

val_in must have size at least the value of ptr_in[n] on the call to mc69_csrlu_convert. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to mc69_csrlu_convert.

ne must be set to the number of entries in the output matrix from the call to mc69_csrlu_convert. If using C indexing, this is the value of ptr_out[n] on exit from mc69_csrlu_convert (for Fortran indexing, subtract 1).

val_out must have size at least the value of ptr_out[n] on exit from mc69_csrlu_convert. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.12.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in full Compressed Sparse Row form, and then converts it to HSL standard format using mc69_csrlu_convert. In addition to the initial conversion, a second set of values matching the same pattern is input. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output:

2.13 Coordinate format

The following routines handle a user-supplied matrix stored in coordinate format. Each non-zero entry in the input matrix is held as a pair (row index, column index) or as a triplet (row index, column index, value). For symmetric, skew symmetric and Hermitian matrices each non-zero entry may be stored as either (i,j) or (j,i) (with appropriate sign or conjugacy). If both entries are input, or if duplicates are input, the values are summed by the routines described in this section.

The triplets are stored using the following data:

`m` is a scalar of type `int` that holds the number of rows of A.

`n` is a scalar of type `int` that holds the number of columns of A.

`ne` is a scalar of type `int` that holds the number of entries of A.

`row` is a rank-one array of type `int`. The first `ne` values `row[j]` must hold the row index for the `j`-th entry of A.

`col` is a rank-one array of type `int`. The first `ne` values `col[j]` must hold the column index for the `j`-th entry of A.

If the values are required in addition to the matrix pattern, the following array is used:

`val` is a rank-one array of package type. The first `ne` values `val[j]` must hold the value for the `j`-th entry of A.

2.13.1 To convert from coordinate format to standard HSL format

To convert a matrix held in coordinate format to standard HSL format, the user may make a call of the following form. This routine checks the user's data and handles duplicate entries (they are summed) and out-of-range entries (they are discarded). For skew-symmetric matrices, diagonal entries are treated as out-of-range entries.

```
int mc69_coord_convert(const int unit, const hsl_matrix_type type,
   const int findex, const int m, const int n, const int ne, const int row_in[],
   const int col_in[], const pkgtype val_in[], int ptr_out[],
   const int lrow, int row_out[], pkgtype val_out[], int *noor,
   int *ndup, int *lmap, int map[]);
```

Arguments:

`unit` holds the Fortran unit number for output. If `unit` $\geq$ `0`, error and warning messages are written to unit `unit`, otherwise they are suppressed. Note that any warning or output messages will refer to the Fortran numbering scheme (as if `findex`$\neq$0), regardless of the value of `findex`.

`type` describes the type of matrix. It must take one of the values described in Section 2.5. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`findex` specifies the findex array index. If the arrays `row_in`, `col_in`, `ptr_out` and `row_out` start numbering at 0 in the C style, `findex` must be set to `0`. If the arrays instead start numbering at 1 in the Fortran style, `findex` must be non-zero. If `findex`=0, an extra copy of `row_in`, `col_in`, `ptr_out` and `row_out` is taken internally by the function.

`m`, `n`, `ne`, `row_in` and `col_in` must be set by the user to hold A in coordinate format, as described in Section 2.13.

`val_in` may be NULL. If not NULL, the first `ne` entries must be set so that `val_in[k]` holds the value of the `k`-th entry of A. If `val_in` is NULL, `val_out` must also be NULL.

lrow specifies the length of row_out and (if it is not NULL) val_out. It must be at least as large as the number of entries in the output matrix. A safe upper bound on this value is the number of entries in the input matrix.

ptr_out and row_out are rank-one arrays. ptr_out is of size n+1 and row_out of size lrow. On exit, they hold A in HSL standard format, as described in Section 2.6.

val_out may be NULL. If not NULL, it should have size at least lrow. On exit the first ptr_out[n] entries will be set such that val_out[k] holds the value of the entry row_out[k]. If val_out is NULL, val_in must also be NULL.

noor contains, on exit, the number of out-of-range entries that were discarded.

ndup contains, on exit, the number of duplicate entries that were summed.

lmap may be NULL. If it is not null then, on entry, *lmap must be the size of the array map[], and on exit it will give the number of entries in map[] that are actually used. If lmap is NULL, map must also be NULL.

map may be NULL. If it is not NULL then it must point to an array of size at least lmap. It should be used if the user wishes to change the values of the entries of A following the call to mc69_coord_convert, and should be passed unaltered to any subsequent calls to mc69_set_values. A detailed description of the output format is given in Section 4.1. If map is NULL, lmap must also be NULL.

Return value:
A successful return indicated by value of 0 or above, with non-zero values indicating a warning. Possible negative values that are associated with an error are:

-1 Allocation error.

-2 Invalid value of type.

-3 m<0 or n<0.

-4 |type| > 1 (square matrix) but m≠n.

-5 ptr[0]<0.

-6 ptr[] is not monotonic increasing.

-10 All entries are out of range.

-11 |type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 type =-3 or -4 (Hermitian case) but one or more entries on the diagonal have non-zero imaginary part.

-15 Only one of val_in and val_out is NULL.

-16 Only one of lmap and map is NULL.

Possible positive values are:

+1 Out-of-range indices found in row_in.

+2 Duplicate indices found in row_in.

+3 Both out-of-range and duplicate entries found.

+4 |type| ≠ 3, 6 and not all entries on the diagonal are present. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

+5 |type| ≠ 3, 6 and not all entries on the diagonal are present and out-of-range and/or duplicate entries found. (Note that no HSL package requires explicit zeros to be on input on the diagonal.)

2.13.2 To set values of A following a conversion

The user may want to change the values of the entries of A following a successful call to mc69_coord_convert. Alternatively, the user may want to include matrix values after a call to mc69_coord_convert with matrix values not not NULL. This can be done by making a call to mc69_set_values, however note that no checks are made on the values of the diagonal entries.

```
void mc69_set_values(const hsl_matrix_type type, const int lmap, const int map[],
  const pkgtype val_in[], const int ne, pkgtype val_out[]);
```

Arguments:

type describes the type of matrix. It must be unchanged since the call to mc69_coord_convert that generated map.

lmap must be unchanged since the call to mc69_coord_convert that generated map.

map must be unchanged since the call to mc69_coord_convert that generated it.

val_in must have size at least the value of ne on the call to mc69_coord_convert. It must be set by the user to hold the new values of the entries of A matching the original matrix that was input to mc69_coord_convert.

ne must be set to the number of entries in the output matrix from the call to mc69_coord_convert. If using C indexing, this is the value of ptr_out[n] on exit from mc69_coord_convert (for Fortran indexing, subtract 1).

val_out must have size at least the value of ptr_out[n] on exit from mc69_coord_convert. On exit, it contains the new values of A in standard HSL format, as described in Section 2.6.

2.13.3 Example

Usage of the routines in this section will be demonstrated using the following matrices

$$A = \begin{pmatrix} 1.0 & 3.0 & & -2.0 \\ 3.0 & 4.0 & 5.0 & \\ & 5.0 & & 6.0 \\ -2.0 & & 6.0 & 7.0+2.0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2.0 & 4.0 & & -3.0 \\ 4.0 & 6.0 & 6.0 & \\ & 6.0 & & 7.0 \\ -3.0 & & 7.0 & 8.0-1.0 \end{pmatrix}.$$

The following code reads a matrix in Coordinate form, and then converts it to HSL standard format using mc69_coord_convert. In addition to the initial conversion, a second set of values matching the same pattern is read. These values are then converted to HSL standard form using mc69_set_values. If provided with the following input (matching the matrices A and B above) on stdin, the code produces the following output.

3 GENERAL INFORMATION

Workspace: HSL_MC69 handles its own memory allocations.

Other routines called directly: None.

Input/output: Error, warning and requested printing only, under control of argument unit in each subroutine call.

Restrictions: m, n, ne ≥ 0; ptr monotonic, ptr(1)=1; type $\in [-6,4] \cup \{6\}$.

Portability: Fortran 95, plus allocatable components of derived types.

4 METHOD

4.1 The format of the map output array

The data stored in map is designed to be easy to apply. It is always stored using 1-based (Fortran) indexing, and consists of two parts:

- The first ptr_out[n] entries specify source locations for each entry of val_out. If map[k] is positive, val_out[k] = val_in[map[k]-1], k = 1,...n. Otherwise, if map[k] is negative, the assignment depends on the type of the matrix:

  Skew symmetric val_out[k] = -val_in[(-map[k])-1];

  Hermitian val_out[k] = conjg(val_in[(-map[k])-1]);

  Otherwise val_out[k] = val_in((-map[k])-1).

- The second part, map[ptr[n]:lmap-1], may be empty. Otherwise entries occur in pairs. Each pair (i, j) = (map[k],map(k+1)), k = ptr[n],ptr[n]+2,... lmap − 1, represents a duplicate that was found. If j is positive then val_out[i-1] = val_out[i-1] + val_in[j-1]. If j is negative and the matrix is Hermitian or skew symmetric, the conjugate or negative value of the val_in[(-j)-1] is used.

Thus, for the simple case where no entries of map(:) are negative, the following code could be used to perform the work of mc69_set_values:

```
for(k=0; k<ptr[n]; k++)
   val_out[k] = val_in[map[k]-1]
for(k=ptr[n]; k<lmap; k+=2)
   val_out[map[k]-1] = val_out[map[k]-1] + val_in[map(k+1)-1]
```

4.2 The routine mc69_cscl_clean

Because the size of the array map depends on the number of duplicates, we make a preliminary pass to count them. To find duplicates quickly, we use a temporary integer array temp that is allocated to have size m and is initialized to zero. When scanning column j, if we find an entry in row i that is within range, we check temp[i]; if it does not have the value j, it is the first occurrence in the column and we then set temp[i] to the value j; otherwise, we have a duplicate.

We take the opportunity in this preliminary scan to count the number of out-of-range entries. To make the sorting in the main scan (slightly) easier, we set row[k] to the artificial value m+1 for each out-of-range entry row[k].

The main pass processes the columns one by one. A heap sort is used to order the entries of each column. This leaves the duplicates next to each other and the out-of-range entries at the end, so a simple scan of the revised column moves all the wanted entries forward so that they are adjacent.

If `val` is not NULL, its entries are permuted during the heap sort and its wanted entries are moved forward and duplicates accumulated during the scan of the column.

If `map` is not NULL, it is allocated before the pass and initialized to represent the identity permutation of the entries by setting `map[k] = k, k = 1,ptr[n]`. It is revised with each data movement made within the sort and the subsequent pass that handles duplicates and out-of-range entries. For each duplicate accumulation, a pair of integers is added at the end of `map`.

Finally, if the matrix is symmetric or Hermitian, the diagonal entries are checked for the relevant properties.

4.3 The routines `mc69_cscl_convert` and `mc69_csru_convert`

Both these routines already have the data in an appropriate format, and merely require the removal of out-of-range and duplicate entries. In the upper CSR case, we exploit the fact that we are only concerned with symmetric, skew-symmetric and Hermitian matrices. In these cases, the pattern of the upper triangle held by rows is identical to the pattern of the lower triangle held by columns, and a simple transform can be applied to obtain the values.

A single pass is made. For each column, first duplicates and out-of-range entries are dropped. Next, entries are sorted into ascending order using a heap sort. Finally, duplicates are identified and removed.

4.4 The routines `mc69_cscu_convert` and `mc69_csrl_convert`

In both these routines we have the transpose of the desired pattern. We proceed in three passes:

1. The first pass (of `row_in`) counts the number of entries in each column of the output matrix. Out-of-range entries are ignored, but duplicates are counted (we cannot detect them at this stage).
2. The second pass (of `row_in`) drops entries into destination locations so that `ptr_out` and `row_out` hold the final output matrix but with duplicates included. By construction, the entries are ordered within each column.
3. The third and final pass (of `row_out`) identifies and sums duplicates to produce the desired matrix.

4.5 The routines `mc69_csclu_convert` and `mc69_csrlu_convert`

Both these routines proceed as `mc69_cscu_convert`, exploiting the availability of the upper triangle to avoid the heap sort required if the lower triangle is used. Entries in the lower triangle are thus ignored (but not counted as out of range). If the number of entries in the lower and upper triangles do not match (after discarding out-of-range entries) an error is issued.

4.6 The routine `mc69_coord_convert`

In this routine, we start with the matrix in coordinate format. We proceed in four passes:

1. The first pass (of `row_in`) counts the number of entries in each column of the output matrix. Out-of-range entries are ignored, but duplicates are counted (we cannot detect them at this stage).
2. The second pass (of `row_in`) drops entries into destination locations so that `ptr_out` and `row_out` hold the final output matrix but with duplicates included. At this stage, the entries in each column are unordered.
3. The third pass (of `row_out`) uses a heap sort to order the entries in each column by increasing row index.
4. The final pass (of `row_out`) identifies and sums duplicates to produce the desired matrix.