# HSL_MC69



## 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 95, plus allocatable components of derived types. **Interfaces:** Fortran, C.

## 2 HOW TO USE THE PACKAGE

### 2.1 Calling sequences

Access to the package requires a `USE` statement

Single precision version

```
USE HSL_MC69_single
```

Double precision version

```
USE HSL_MC69_double
```

Complex version

```
USE HSL_MC69_complex
```

Double complex version

```
USE HSL_MC69_double_complex
```

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.

| Input format | matrix type | Page |
|---|---|---|
| Compressed sparse column | All | 8 |
| Upper compressed sparse column | Symmetric, skew symmetric, Hermitian | 16 |
| Full compressed sparse column | Symmetric, skew symmetric, Hermitian | 21 |
| Compressed sparse row | All | 26 |
| Upper compressed sparse row | Symmetric, skew symmetric, Hermitian | 31 |
| Compressed sparse row | Symmetric, skew symmetric, Hermitian | 36 |
| Coordinate | All | 41 |

### 2.2 Argument lists and calling sequences

#### 2.2.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments, which are always at the end of the argument list. Since we reserve the right to modify the argument list and to add additional optional arguments in future releases of the code, **we strongly recommend that all optional arguments be called by keyword, not by position**.

#### 2.2.2 Integer, real and package types

INTEGER denotes default integer and INTEGER(long) denotes INTEGER(kind=selected_int_kind(18)).
REAL denotes default real if the single precision version or the complex version is being used, and double precision real if the double precision or double precision complex version is being used.
We use the term **package type** to mean default real if the single precision version is being used, double precision real for the double precision version, default complex for the complex version and double precision complex for the double complex version.

### 2.3 Matrix type constants

The following INTEGER parameters are defined:

| | | |
|---|---|---|
| HSL_MATRIX_UNDEFINED | = 0 | undefined/unknown |
| | | |
| 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 |
| | | |
| HSL_MATRIX_CPLX_RECT | = -1 | complex rectangular |
| HSL_MATRIX_CPLX_UNSYMMETRIC | = -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 |

### 2.4 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 `INTEGER` scalar that holds the number of rows in $A$.

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

`ptr` is a rank-one `INTEGER` array of size `n+1`. `ptr(j)` must hold the position in `row` of the first entry in column `j` and `ptr(n+1)` must be one more than the total number of entries.

`row` is a rank-one `INTEGER` array. The first `ptr(n+1)-1` 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. 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 array of package type. `val(k)` must hold the value of the entry in `row(k)`.

#### 2.4.1 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.

```
call mc69_verify(lp, matrix_type, m, n, ptr, row, flag, more [,val])
```

`lp` is an `INTENT(IN)` scalar of type `INTEGER`. If `lp` $\geq$ `0`, error messages are printed on unit `lp`.

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of the matrix. It must have one of the values given in Section 2.3. 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).

`m`, `n`, `ptr` and `row` are of `INTENT(IN)` and must be set by the user to hold $A$ in standard HSL format as described in Section 2.4.

`flag` is a scalar `INTENT(OUT)` of type `INTEGER`. On exit, a value of 0 indicates the matrix is in standard HSL format. Negative values are associated with an error; see Section 2.4.3 for details.

`more` is a scalar `INTENT(OUT)` of type `INTEGER`. If `flag` has a negative value on exit, `more` may provide further information; see Section 2.4.3.

`val` is an optional `INTENT(IN)` rank-one array of package type. If present, `val(k)` must hold the value of the entry in `row(k)`.

### 2.4.2 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 make the following call:

```
call mc69_print(lp, lines, matrix_type, m, n, ptr, row[, val])
```

`lp` is an `INTENT(IN)` scalar of type `INTEGER`. If `lp` $\geq$ `0`, the matrix is printed on unit `lp`. There is an immediate return if `lp` $<$ `0`.

`lines` is an `INTENT(IN)` scalar of type `INTEGER`. If **`lines`** $\geq 0$, a summary of the matrix will be printed of not more than `lines` lines. Otherwise, the whole matrix will be printed.

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of the matrix. It must have one of the values given in Section 2.3.

`m`, `n`, `ptr` and `row` are of `INTENT(IN)` and must be set by the user to hold *A* in standard HSL format as described in Section 2.4.

`val` is an optional `INTENT(IN)` rank-one array of package type. If present, it must be of size `ptr(n+1)-1` and `val(k)` must hold the value of the entry in `row(k)`.

### 2.4.3 Return codes

Possible negative values of `flag` that may be returned by `mc69_verify` are:

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

`-2` Invalid value of `matrix_type`.

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

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

`-5` `ptr(1)`<`1`. `more` is set to `ptr(1)`.

`-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` |`matrix_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` `matrix_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.

### 2.4.4 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`.

```
program hsl_mc69ds
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: matrix_type, m, n, flag, more
   integer, dimension(:), allocatable :: ptr, row
   real(wp), dimension(:), allocatable :: val

   ! Read matrix in HSL standard form
   read(*, "(3i8)") matrix_type, m, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   ! Verify matrix is in correct form
   call mc69_verify(6, matrix_type, m, n, ptr, row, flag, more, val=val)
   if(flag.ne.0) then
      write(*, "(a,i3,a,i5)") &
         "Error return from mc69_verify with flag = ", flag, ", more = ", more
      stop
   endif

   ! Print out pattern in no more than 10 lines
   write(*, "(a)") "Matrix pattern:"
   call mc69_print(6, 10, matrix_type, m, n, ptr, row)

   ! Print out matrix values in no more than 10 lines
   write(*, "(/a)") "Matrix values:"
   call mc69_print(6, 10, matrix_type, m, n, ptr, row, val=val)
end program hsl_mc69ds
```

If provided with the following input (matching the matrix $A$ above),

```
       4       4       4
       1       4       6       7       8
       1       2       4       2       3       4
       4
```

the code produces the following output.

```
  1.0000E+00  3.0000E+00 -2.0000E+00  4.0000E+00
  5.0000E+00  6.0000E+00  7.0000E+00
Matrix pattern:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1: x x   x
2: x x x
3:   x   x
4: x   x x

Matrix values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00              -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00               6.0000E+00
4:  -2.0000E+00                6.0000E+00   7.0000E+00
```

### 2.5 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 `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 `n` values must be set such that `ptr(j)` holds the position in `row` of the first entry in column `j` and `ptr(n+1)` must be one more than the total number of entries.

`row` is a rank-one array of type `INTEGER`. The first `ptr(n+1)-1` entries hold the row indices of the entries, with the row indices for the entries in column 1 preceding those for column 2, 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)`.

#### 2.5.1 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.

```
call mc69_cscl_clean(matrix_type, m, n, ptr, row, flag[, val, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`m`, `n` are of `INTENT(IN)`, `ptr` and `row` are of `INTENT(INOUT)` and must be set by the user to hold $A$ in compressed sparse column format, as described in Section 2.5. On exit, `ptr` and `row` will have been modified to hold the matrix in standard HSL format.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.5.4 for details.

`val` is an optional `INTENT(INOUT)` rank-one array of package type. If present, on input the first `ptr(n+1)-1` 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`.

`lmap` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If `lmap` is present, `map` must also be present.

`map` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of type `INTEGER`. It should be present 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 present, `lmap` must also be present.

`lp` is an optional `INTENT(IN)` scalar of type `INTEGER`. If present and `lp` $\geq$ `0`, error and warning messages are written to unit `lp`.

`noor` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If present, on exit it contains the number of out-of-range entries that were discarded.

`ndup` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If present, on exit it contains the number of duplicate entries that were summed.

### 2.5.2 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.

```
call mc69_cscl_convert(matrix_type, m, n, ptr_in, row_in, ptr_out, row_out, &
   flag[, val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`m`, `n`, `ptr_in` and `col_in` are of `INTENT(IN)` and must be set by the user to hold *A* in compressed sparse column format, as described in Section 2.5.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold *A* in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.5.4 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, on input the first `ptr_in(n+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `row_in(k)`. If `val_in` is present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one allocatable array of package type. If present, on exit it will be allocated and the first `ptr_out(n+1)-1` entries will be set such that `val_out(k)` holds the value of the entry `row_out(k)`. If `val_out` is present, `val_in` must also be present.

`lmap` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If `lmap` is present, `map` must also be present.

`map` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of type `INTEGER`. It should be present 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 present, `lmap` must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and lp $\geq$ 0, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

### 2.5.3 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 with matrix values not present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_cscl_clean or mc69_cscl_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It must have size at least the value of ptr(n+1)-1 on the call to mc69_cscl_clean (or ptr_in(n+1)-1 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr(n+1)-1 on exit from mc69_cscl_clean or mc69_cscl_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr(n+1)-1 on exit from mc69_cscl_clean (or ptr_out(n+1)-1 on exit from mc69_cscl_convert). On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

### 2.5.4 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 matrix_type.

-3 m<0 or n<0.

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

-5 ptr(1)<1.

-6 ptr(1:n+1) is not monotonic increasing.

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

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

-12 matrix_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 present.

-16 Only one of lmap and map is present.

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 |matrix_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 |matrix_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.)

### 2.5.5 Example

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

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

The following code reads a matrix in Compressed Sparse Column form, and then performs an *in-place* conversion to HSL standard form using mc69_cscl_clean.

```
program hsl_mc69ds1
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, m, n, flag
   integer, dimension(:), allocatable :: ptr, row
   real(wp), dimension(:), allocatable :: val

   ! Read matrix in CSC form
   read(*, "(3i8)") matrix_type, m, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)
```

```
   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form in place
   call mc69_cscl_clean(matrix_type, m, n, ptr, row, flag, val=val)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_cscl_clean with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix values in no more than 10 lines
   write(*, "()")
   call mc69_print(6, 10, matrix_type, m, n, ptr, row, val=val)
end program hsl_mc69ds1
```

If provided with the following input (matching the matrix *A* above),

```
       4       4       4
       1       4       6       7       9
       1       4       2       2       3       4
       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  4.0000E+00
  5.0000E+00  6.0000E+00  7.0000E+00  2.0000E+00
```

the code produces the following output.

```
Input:
Column  1:     1 (    1.00E+00)     4 (   -2.00E+00)     2 (    3.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    7.00E+00)     4 (    2.00E+00)

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
```

```
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00
```

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`.

```
program hsl_mc69ds2
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, m, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out

   ! Read matrix in CSC form
   read(*, "(3i8)") matrix_type, m, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_cscl_convert(matrix_type, m, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_cscl_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
```

```
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds2
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4       4
       1       4       6       7       9
       1       4       2       2       3       4
       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  4.0000E+00
  5.0000E+00  6.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00 -3.0000E+00  4.0000E+00  6.0000E+00
  6.0000E+00  7.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Column  1:     1 (    1.00E+00)     4 (   -2.00E+00)     2 (    3.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    7.00E+00)     4 (    2.00E+00)

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00               -3.0000E+00
```

```
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00                7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

### 2.6 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 `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+1)` must be one more than the total number of entries.

`row` is a rank-one array of type `INTEGER`. The first `ptr(n+1)-1` entries hold the row indices of the entries in the **upper triangle** of *A*, with the row indices for the entries in column 1 preceding those for column 2, 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)`.

#### 2.6.1 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.

```
call mc69_cscu_convert(matrix_type, n, ptr_in, row_in, ptr_out, row_out, &
   flag[, val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3 for a symmetric, skew-symmetric or Hermitian matrix.

`n`, `ptr_in` and `row_in` are of `INTENT(IN)` and must be set by the user to hold *A* in upper compressed sparse row format, as described in Section 2.6.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold *A* in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.6.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, on input the first `ptr_in(n+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `row_in(k)`. If `val_in` is present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one allocatable array of package type. If present, on exit it will be allocated and the first `ptr_out(n+1)-1` entries will be set such that `val_out(k)` holds the value of the entry `row_out(k)`. If `val_out` is present, `val_in` must also be present.

`lmap` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If `lmap` is present, `map` must also be present.

map is an optional INTENT(OUT) rank-one ALLOCATABLE array of type INTEGER. It should be present 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 present, lmap must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and lp $\geq$ 0, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

### 2.6.2 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 present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_cscu_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_cscu_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_cscu_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It must have size at least the value of ptr_in(n+1)-1 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr_out(n+1)-1 on exit from mc69_cscu_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr_out(n+1)-1 on exit from mc69_cscu_convert. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

### 2.6.3 Return codes

A successful return from a call to mc69_cscu_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 matrix_type.

-3 n<0.

-5 ptr(1)<1.

-6 ptr(1:n+1) is not monotonic increasing.

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

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

-12 matrix_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 present.

-16 Only one of lmap and map is present.

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 |matrix_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 |matrix_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.)

### 2.6.4 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.

```
program hsl_mc69ds3
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out

   ! Read symmetric matrix in upper CSC form
   read(*, "(2i8)") matrix_type, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
```

```
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_cscu_convert(matrix_type, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_cscl_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds3
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4
       1       2       4       5       9
       1       1       2       2       3       1
       4       4
  1.0000E+00  3.0000E+00  4.0000E+00  5.0000E+00
```

```
  6.0000E+00 -2.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00  4.0000E+00  6.0000E+00  6.0000E+00
  7.0000E+00 -3.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Column  1:     1 (    1.00E+00)
Column  2:     1 (    3.00E+00)     2 (    4.00E+00)
Column  3:     2 (    5.00E+00)
Column  4:     3 (    6.00E+00)     1 (   -2.00E+00)     4 (    7.00E+00)     4 (    2.00E+00)

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00               -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00                7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

### 2.7 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+1)` must be one more than the total number of entries.

`row` is a rank-one array of type `INTEGER`. The first `ptr(n+1)-1` 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)`.

#### 2.7.1 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.

```
call mc69_csclu_convert(matrix_type, n, ptr_in, row_in, ptr_out, row_out, flag[, &
   val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3 for a symmetric, skew-symmetric or Hermitian matrix.

`n`, `ptr_in` and `row_in` are of `INTENT(IN)` and must be set by the user to hold $A$ in full compressed sparse column format, as described in Section 2.7.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold $A$ in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.7.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, the first `ptr_in(n+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `row_in(k)`. If `val_in` is present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of package type. 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 present, `val_in` must also be present.

lmap is an optional INTENT(OUT) scalar of type INTEGER. If lmap is present, map must also be present.

map is an optional INTENT(OUT) rank-one ALLOCATABLE array of type INTEGER. It should be present 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 present, lmap must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and lp $\geq$ 0, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

### 2.7.2 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 present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_csclu_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_csclu_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_csclu_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It must have size at least the value of ptr_in(n+1)-1 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr_out(n+1)-1 on exit from mc69_csclu_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr_out(n+1)-1 on exit from mc69_csclu_convert. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

### 2.7.3 Return codes

A successful return from a call to mc69_csclu_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 matrix_type.

-3 n<0.

-5 ptr(1)<1.

-6 ptr(1:n+1) 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 |matrix_type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 matrix_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 present.

-16 Only one of lmap and map is present.

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 |matrix_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 |matrix_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.)

#### 2.7.4 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.

```
program hsl_mc69ds4
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out
```

```
   ! Read matrix in full CSC form
   read(*, "(2i8)") matrix_type, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(1x,i4,1x,'(',es9.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_csclu_convert(matrix_type, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_csclu_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(1x,i4,1x,'(',es9.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds4
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4
       1       4       7       9      13
```

```
      1       4       2       1       2       3
      2       4       1       3       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  3.0000E+00
  4.0000E+00  5.0000E+00  5.0000E+00  6.0000E+00
 -2.0000E+00  7.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00 -3.0000E+00  4.0000E+00  4.0000E+00
  6.0000E+00  6.0000E+00  6.0000E+00  7.0000E+00
 -3.0000E+00  7.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Column  1:    1 ( 1.00E+00)    4 (-2.00E+00)    2 ( 3.00E+00)
Column  2:    1 ( 3.00E+00)    2 ( 4.00E+00)    3 ( 5.00E+00)
Column  3:    2 ( 5.00E+00)    4 ( 6.00E+00)
Column  4:    1 (-2.00E+00)    3 ( 7.00E+00)    4 ( 7.00E+00)    4 ( 2.00E+00)

Output:
Column  1:    1 ( 1.00E+00)    2 ( 3.00E+00)    4 (-2.00E+00)
Column  2:    2 ( 4.00E+00)    3 ( 5.00E+00)
Column  3:    4 ( 7.00E+00)
Column  4:    4 ( 9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00              -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00               7.0000E+00
4:  -2.0000E+00               7.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00              -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00               7.0000E+00
4:  -3.0000E+00               7.0000E+00   7.0000E+00
```

### 2.8 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+1)` must be one more than the total number of entries.

`col` is a rank-one array of type `INTEGER`. The first `ptr(m+1)-1` entries hold the column indices of the entries in $A$, with the column indices for the entries in row 1 preceding those for row 2, 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)`.

#### 2.8.1 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.

```
call mc69_csrl_convert(matrix_type, m, n, ptr_in, col_in, ptr_out, row_out, &
   flag[, val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`m`, `n`, `ptr_in` and `col_in` are of `INTENT(IN)` and must be set by the user to hold $A$ in compressed sparse row format, as described in Section 2.8.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold $A$ in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.8.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, on input the first `ptr_in(m+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `col_in(k)`. If `val_in` is present, `val_out` must also be present.

val_out is an optional INTENT(OUT) rank-one allocatable array of package type. If present, on exit it will be allocated and the first ptr_out(n+1)-1 entries will be set such that val_out(k) holds the value of the entry row_out(k). If val_out is present, val_in must also be present.

lmap is an optional INTENT(OUT) scalar of type INTEGER. If lmap is present, map must also be present.

map is an optional INTENT(OUT) rank-one ALLOCATABLE array of type INTEGER. It should be present 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 present, lmap must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and $\text{lp} \geq 0$, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

### 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_csrl_convert. Alternatively, the user may want to include matrix values after a call to mc69_csrl_convert with matrix values not present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_csrl_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_csrl_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_csrl_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It must have size at least the value of ptr_in(m+1)-1 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr_out(n+1)-1 on exit from mc69_csrl_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr_out(n+1)-1 on exit from mc69_csrl_convert. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

#### 2.8.3 Return codes

A successful return from a call to `mc69_csrl_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 `matrix_type`.

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

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

`-5` `ptr(1)`$<$`1`.

`-6` `ptr(1:n+1)` is not monotonic increasing.

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

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

`-12` `matrix_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 present.

`-16` Only one of `lmap` and `map` is present.

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` $|$`matrix_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` $|$`matrix_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.)

#### 2.8.4 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`.

```
program hsl_mc69ds5
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, m, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out

   ! Read matrix in CSR form
   read(*, "(3i8)") matrix_type, m, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, m
      write(*, "(a,i2,':')",advance="no") "Row ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_csrl_convert(matrix_type, m, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_csrl_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)
```

```
   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds5
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4       4
       1       2       4       5       9
       1       1       2       2       3       1
       4       4
  1.0000E+00  3.0000E+00  4.0000E+00  5.0000E+00
  6.0000E+00 -2.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00  4.0000E+00  6.0000E+00  6.0000E+00
  7.0000E+00 -3.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Row  1:     1 (    1.00E+00)
Row  2:     1 (    3.00E+00)     2 (    4.00E+00)
Row  3:     2 (    5.00E+00)
Row  4:     3 (    6.00E+00)     1 (   -2.00E+00)     4 (    7.00E+00)     4 (    2.00E+00)

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00               -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00                7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

### 2.9 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 `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 `col` of the first entry in row `j` and `ptr(n+1)` must be one more than the total number of entries.

`col` is a rank-one array of type `INTEGER`. The first `ptr(n+1)-1` entries hold the column indices of the entries in $A$, with the column indices for the entries in row 1 preceding those for row 2, 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)`.

#### 2.9.1 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.

```
call mc69_csru_convert(matrix_type, n, ptr_in, col_in, ptr_out, row_out, &
   flag[, val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3 for symmetric, skew-symmetric or Hermitian matrices.

`n`, `ptr_in` and `col_in` are of `INTENT(IN)` and must be set by the user to hold $A$ in upper compressed sparse row format, as described in Section 2.9.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold $A$ in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.9.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, on input the first `ptr_in(n+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `col_in(k)`. If `val_in` is present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one allocatable array of package type. If present, on exit it will be allocated and the first `ptr_out(n+1)-1` entries will be set such that `val_out(k)` holds the value of the entry `row_out(k)`. If `val_out` is present, `val_in` must also be present.

`lmap` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If `lmap` is present, `map` must also be present.

`map` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of type `INTEGER`. It should be present 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 present, `lmap` must also be present.

`lp` is an optional `INTENT(IN)` scalar of type `INTEGER`. If present and `lp` $\geq$ `0`, error and warning messages are written to unit `lp`.

`noor` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If present, on exit it contains the number of out-of-range entries that were discarded.

`ndup` is an optional `INTENT(OUT)` scalar of type `INTEGER`. If present, on exit it contains the number of duplicate entries that were summed.

#### 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_csru_convert`. Alternatively, the user may want to include matrix values after a call to `mc69_csru_convert` with matrix values not present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must be unchanged since the call to `mc69_csru_convert` that generated `map`.

`lmap` is an `INTENT(IN)` scalar of type `INTEGER` that must be unchanged since the call to `mc69_csru_convert` that generated `map`.

`map` is an `INTENT(IN)` rank-one array of type `INTEGER` that must be unchanged since the call to `mc69_csru_convert` that generated it.

`val_in` is an `INTENT(IN)` rank-one array of package type. It must have size at least the value of `ptr_in(n+1)-1` 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` is an `INTENT(IN)` scalar argument of type `INTEGER` that must be set to the value of `ptr_out(n+1)-1` on exit from `mc69_csru_convert`.

`val_out` is an `INTENT(OUT)` rank-one array of package type. It must have size at least the value of `ptr_out(n+1)-1` on exit from `mc69_csru_convert`. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

#### 2.9.3 Return codes

A successful return from a call to `mc69_csru_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 `matrix_type`.

`-3` `n<0`.

-5 `ptr(1)`<1.

-6 `ptr(1:n+1)` is not monotonic increasing.

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

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

-12 `matrix_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 present.

-16 Only one of `lmap` and `map` is present.

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 |`matrix_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 |`matrix_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.)

### 2.9.4 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 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`.

```
program hsl_mc69ds6
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out

   ! Read symmetric matrix in upper CSR form
   read(*, "(2i8)") matrix_type, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
```

```
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Row ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_csru_convert(matrix_type, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_csrl_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds6
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4
       1       4       6       7       9
       1       4       2       2       3       4
       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  4.0000E+00
```

```
  5.0000E+00  6.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00 -3.0000E+00  4.0000E+00  6.0000E+00
  6.0000E+00  7.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Row  1:     1 (    1.00E+00)     4 (   -2.00E+00)     2 (    3.00E+00)
Row  2:     2 (    4.00E+00)     3 (    5.00E+00)
Row  3:     4 (    6.00E+00)
Row  4:     4 (    7.00E+00)     4 (    2.00E+00)

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00              -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00               6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00              -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00               7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

### 2.10 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 `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 `col` of the first entry in row `j` and `ptr(n+1)` must be one more than the total number of entries.

`col` is a rank-one array of type `INTEGER`. The first `ptr(n+1)-1` 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)`.

#### 2.10.1 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.

```
call mc69_csrlu_convert(matrix_type, n, ptr_in, col_in, ptr_out, row_out, flag[, val_in, &
   val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3 for symmetric, skew-symmetric or Hermitian matrices.

`n`, `ptr_in` and `col_in` are of `INTENT(IN)` and must be set by the user to hold $A$ in full compressed sparse column format, as described in Section 2.10.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold $A$ in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.10.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, the first `ptr_in(n+1)-1` entries must be set so that `val_in(k)` holds the value of the entry `row_in(k)`. If `val_in` is present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of package type. 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 present, `val_in` must also be present.

lmap is an optional INTENT(OUT) scalar of type INTEGER. If lmap is present, map must also be present.

map is an optional INTENT(OUT) rank-one ALLOCATABLE array of type INTEGER. It should be present 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 present, lmap must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and lp $\geq$ 0, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

### 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_csrlu_convert. Alternatively, the user may want to include matrix values after a call to mc69_csrlu_convert with matrix values not present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_csrlu_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_csrlu_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_csrlu_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It must have size at least the value of ptr_in(n+1)-1 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr_out(n+1)-1 on exit from mc69_csrlu_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr_out(n+1)-1 on exit from mc69_csrlu_convert. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

### 2.10.3 Return codes

A successful return from a call to mc69_csrlu_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 matrix_type.

-3 n<0.

-5 ptr(1)<1.

-6 ptr(1:n+1) 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 |matrix_type| =3 (positive-definite case) but one or more diagonal entries are not positive.

-12 matrix_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 present.

-16 Only one of lmap and map is present.

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 |matrix_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 |matrix_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.)

### 2.10.4 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 read. These values are then converted to HSL standard form using mc69_set_values.

```
program hsl_mc69ds7
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, n, flag, lmap
   integer, dimension(:), allocatable :: ptr, row, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out
```

```
   ! Read matrix in full CSR form
   read(*, "(2i8)") matrix_type, n
   allocate(ptr(n+1)); read(*, "(6i8)") ptr(:)
   allocate(row(ptr(n+1)-1)); read(*, "(6i8)") row(:)
   allocate(val(ptr(n+1)-1)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Row ", i
      do j = ptr(i), ptr(i+1)-1
         write(*, "(1x,i4,1x,'(',es9.2,')')", advance="no") row(j), val(j)
      end do
      write(*, "()")
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_csrlu_convert(matrix_type, n, ptr, row, ptr_out, row_out, flag, &
      val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_csrlu_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(1x,i4,1x,'(',es9.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, n, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds7
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4
       1       4       7       9      13
```

```
      1       4       2       1       2       3
      2       4       1       3       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  3.0000E+00
  4.0000E+00  5.0000E+00  5.0000E+00  6.0000E+00
 -2.0000E+00  7.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00 -3.0000E+00  4.0000E+00  4.0000E+00
  6.0000E+00  6.0000E+00  6.0000E+00  7.0000E+00
 -3.0000E+00  7.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
Row  1:    1 ( 1.00E+00)    4 (-2.00E+00)    2 ( 3.00E+00)
Row  2:    1 ( 3.00E+00)    2 ( 4.00E+00)    3 ( 5.00E+00)
Row  3:    2 ( 5.00E+00)    4 ( 6.00E+00)
Row  4:    1 (-2.00E+00)    3 ( 7.00E+00)    4 ( 7.00E+00)    4 ( 2.00E+00)

Output:
Column  1:    1 ( 1.00E+00)    2 ( 3.00E+00)    4 (-2.00E+00)
Column  2:    2 ( 4.00E+00)    3 ( 5.00E+00)
Column  3:    4 ( 7.00E+00)
Column  4:    4 ( 9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                7.0000E+00
4:  -2.0000E+00                7.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00               -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00                7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

### 2.11 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 `INTEGER` that holds the number of rows of $A$.

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

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

`row` is a rank-one array of type `INTEGER`. 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 `INTEGER`. 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 $A$.

#### 2.11.1 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.

```
call mc69_coord_convert(matrix_type, m, n, ne, row, col, ptr_out, row_out, flag[, &
   val_in, val_out, lmap, map, lp, noor, ndup])
```

`matrix_type` is an `INTENT(IN)` scalar of type `INTEGER` that describes the type of matrix. It must take one of the values described in Section 2.3. If this argument has value `0` (`HSL_MATRIX_UNDEFINED`), the matrix will be treated as if it were rectangular.

`m`, `n`, `ne`, `row_in` and `col_in` are of `INTENT(IN)` and must be set by the user to hold $A$ in coordinate format, as described in Section 2.11.

`ptr_out` and `row_out` are `INTENT(OUT)` rank-one arrays of type `INTEGER`. `ptr_out` is of size `n+1` and `row_out` is allocatable. On exit, they hold $A$ in HSL standard format, as described in Section 2.4.

`flag` is an `INTENT(OUT)` scalar of type `INTEGER`. 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 2.11.3 for details.

`val_in` is an optional `INTENT(IN)` rank-one array of package type. If present, 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 present, `val_out` must also be present.

`val_out` is an optional `INTENT(OUT)` rank-one `ALLOCATABLE` array of package type. 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 present, `val_in` must also be present.

lmap is an optional INTENT(OUT) scalar of type INTEGER. If lmap is present, map must also be present.

map is an optional INTENT(OUT) rank-one ALLOCATABLE array of type INTEGER. It should be present 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 present, lmap must also be present.

lp is an optional INTENT(IN) scalar of type INTEGER. If present and lp $\geq$ 0, error and warning messages are written to unit lp.

noor is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of out-of-range entries that were discarded.

ndup is an optional INTENT(OUT) scalar of type INTEGER. If present, on exit it contains the number of duplicate entries that were summed.

#### 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_coord_convert. Alternatively, the user may want to include matrix values after a call to mc69_coord_convert with matrix values not present. 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.

```
call mc69_set_values(matrix_type, lmap, map, val_in, ne, val_out)
```

matrix_type is an INTENT(IN) scalar of type INTEGER that describes the type of matrix. It must be unchanged since the call to mc69_coord_convert that generated map.

lmap is an INTENT(IN) scalar of type INTEGER that must be unchanged since the call to mc69_coord_convert that generated map.

map is an INTENT(IN) rank-one array of type INTEGER that must be unchanged since the call to mc69_coord_convert that generated it.

val_in is an INTENT(IN) rank-one array of package type. It 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 is an INTENT(IN) scalar argument of type INTEGER that must be set to the value of ptr_out(n+1)-1 on exit from mc69_coord_convert.

val_out is an INTENT(OUT) rank-one array of package type. It must have size at least the value of ptr_out(n+1)-1 on exit from mc69_coord_convert. On exit, it contains the new values of *A* in standard HSL format, as described in Section 2.4.

#### 2.11.3 Return codes

A successful return from a call to mc69_coord_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 matrix_type.

-3 m<0 or n<0.

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

-10 All entries are out of range.

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

-12 matrix_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 present.

-16 Only one of lmap and map is present.

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 |matrix_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 |matrix_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.)

### 2.11.4 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 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.

```
program hsl_mc69ds8
   use hsl_mc69_double
   implicit none

   integer, parameter :: wp = kind(0d0)

   integer :: i, j, matrix_type, m, n, ne, flag, lmap
   integer, dimension(:), allocatable :: row, col, ptr_out, row_out, map
   real(wp), dimension(:), allocatable :: val, val_out

   ! Read matrix in coordinate form
   read(*, "(4i8)") matrix_type, m, n, ne
   allocate(row(ne)); read(*, "(6i8)") row(:)
```

```
   allocate(col(ne)); read(*, "(6i8)") col(:)
   allocate(val(ne)); read(*, "(4es12.4)") val(:)

   write(*, "(a)") "Input:"
   do i = 1, ne
      write(*, "(a,2i4,es12.2)") "row, col, val = ", row(i), col(i), val(i)
   end do

   ! Convert to HSL standard form out of place
   allocate(ptr_out(n+1))
   call mc69_coord_convert(matrix_type, m, n, ne, row, col, ptr_out, row_out, &
      flag, val_in=val, val_out=val_out, lmap=lmap, map=map)
   if(flag.lt.0) then
      write(*, "(a,i3)") &
         "Error return from mc69_coord_convert with flag = ", flag
      stop
   endif

   write(*, "(/a)") "Output:"
   do i = 1, n
      write(*, "(a,i2,':')",advance="no") "Column ", i
      do j = ptr_out(i), ptr_out(i+1)-1
         write(*, "(2x,i4,1x,'(',es12.2,')')", advance="no") &
            row_out(j), val_out(j)
      end do
      write(*, "()")
   end do

   ! Print out matrix
   write(*, "()")
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)

   ! Read and apply new values
   read(*, "(4es12.4)") val(:)
   call mc69_set_values(matrix_type, lmap, map, val, ptr_out(n+1)-1, val_out)
   write(*, "(/a)") "After applying new values:"
   call mc69_print(6, 0, matrix_type, m, n, ptr_out, row_out, val=val_out)
end program hsl_mc69ds8
```

If provided with the following input (matching the matrices *A* and *B* above),

```
       4       4       4       8
       1       1       1       2       2       4
       4       4
       1       4       2       2       3       3
       4       4
  1.0000E+00 -2.0000E+00  3.0000E+00  4.0000E+00
  5.0000E+00  6.0000E+00  7.0000E+00  2.0000E+00
  2.0000E+00 -3.0000E+00  4.0000E+00  6.0000E+00
  6.0000E+00  7.0000E+00  8.0000E+00 -1.0000E+00
```

the code produces the following output.

```
Input:
row, col, val =    1   1    1.00E+00
row, col, val =    1   4   -2.00E+00
row, col, val =    1   2    3.00E+00
row, col, val =    2   2    4.00E+00
row, col, val =    2   3    5.00E+00
row, col, val =    4   3    6.00E+00
row, col, val =    4   4    7.00E+00
row, col, val =    4   4    2.00E+00

Output:
Column  1:     1 (    1.00E+00)     2 (    3.00E+00)     4 (   -2.00E+00)
Column  2:     2 (    4.00E+00)     3 (    5.00E+00)
Column  3:     4 (    6.00E+00)
Column  4:     4 (    9.00E+00)

Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   1.0000E+00   3.0000E+00               -2.0000E+00
2:   3.0000E+00   4.0000E+00   5.0000E+00
3:                5.0000E+00                6.0000E+00
4:  -2.0000E+00                6.0000E+00   9.0000E+00

After applying new values:
Real symmetric indefinite matrix, dimension 4x4 with 7 entries.
1:   2.0000E+00   4.0000E+00               -3.0000E+00
2:   4.0000E+00   6.0000E+00   6.0000E+00
3:                6.0000E+00                7.0000E+00
4:  -3.0000E+00                7.0000E+00   7.0000E+00
```

## 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 lp in each subroutine call.

**Restrictions:** m, n, ne $\geq$ 0; ptr monotonic, ptr(1)=1; matrix_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 consists of two parts:

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

  **Skew symmetric** val_out(k) = -val_in(-map(k));

  **Hermitian** val_out(k) = conjg(val_in(-map(k)));

  **Otherwise** val_out(k) = val_in(-map(k)).

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

```
do k = 1, ptr(n+1)-1
   val_out(k) = val_in(map(k))
end do
do k = ptr(n+1), lmap, 2
   val_out(map(k)) = val_out(map(k)) + val_in(map(k+1))
end do
```

### 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 present, 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 present, 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+1)-1`. 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.