Science and Technology Facilities Council

HSL_MC78

C INTERFACE HSL

1 SUMMARY

Given an elimination order, HSL_MC78 performs common tasks required in the **analyse phase of a symmetric sparse direct solver**. Either the entire analyse may be performed or individual tasks. No checking is performed on the validity of user data, and failure to supply valid data will result in undefined behaviour.

Given the sparsity pattern of a sparse symmetric matrix A and permutation P, HSL_MC78 finds the pattern of the Cholesky factor L such that $PAP^{-1} = LL^T$. The pattern of A may be provided in either assembled or elemental format. The permutation P is referred to as the *elimination (pivot) order*.

Assembled matrices are specified by listing, for each column of A, the indices for rows of that column that are non-zero. *Elemental matrices* are formed as the sum $\sum_{k=1}^{m} A^{(k)}$ of element matrices, where only the rows and columns of $A^{(k)}$ that correspond to variables in the kth element are non-zero.

To reduce the amount of matrix data read during the analysis, supervariables of A may be identified. A *supervariable* is a set of columns of A that have the same sparsity pattern.

An *elimination tree* is built that describes the structure of the Cholesky factor in terms of data dependence between pivotal columns. This allows permutations of the elimination order that do not affect the number of entries in L to be identified and allows fast algorithms to be used in determining the exact structure of L.

A *supernode* is a set of columns that have the same pattern in the matrix L. This pattern is stored as a single row list for each supernode. The condensed version of the elimination tree consisting of supernodes is referred to as the *assembly tree*. To increase efficiency in a subsequent factorization phase, supernodes may be merged through a supernode amalgamation heuristic.

HSL_MC78 supports the use of 2×2 and larger block pivots. They must be input as consecutive pivots in the elimination order. The identification and use of supervariables of A is also optionally supported, allowing the matrix data to be compressed giving a consequent increase in performance for some problems. However, the simultaneous use of supervariables and block pivots is not currently supported.

ATTRIBUTES — Version: 1.6.1 (15 April 2023). **Interfaces:** C, Fortran. **Types:** Integer, Long integer. **Original date:** October 2010. **Origin:** J. D. Hogg, Rutherford Appleton Laboratory. **Language:** Fortran 2008 subset (F95 + TR15581 + C interoperability).

2 HOW TO USE THE PACKAGE

2.1 C interface to Fortran code

This package is written in Fortran and a wrapper is provided for C programmers. This wrapper may only implement a subset of the full functionality described in the Fortran user documentation.

The subroutines mc78_analyse_asm and mc78_analyse_elt **only** will automatically convert between 0-based (C) and 1-based (Fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing. The conversion may be disabled by setting the control parameter control.f_arrays=1 (i.e. true) and supplying all data using 1-based indexing. With 0-based indexing, the matrix is treated as having rows and columns $0, 1, \ldots n-1$. In this document, we assume 1-based indexing when referencing the matrix, except in the description of mc78_analyse_asm and mc78_analyse_elt.

The wrapper uses the Fortran 2003 interoperability features in addition to the C_SIZEOF function defined by Fortran 2008. **Matching C and Fortran compilers must be used**, for example, gcc and gfortran, or icc and ifort. If

the Fortran compiler is not used to link the user's program, additional Fortran compiler libraries may need to be linked explicitly.

Access to the package requires inclusion of a header file

Integer version

```
#include "hsl_mc78i.h"
```

Long version

```
#include "hsl_mc78l.h"
```

If it is required to use more than one module at the same time, the postfix _i (integer) or _l (long) should be appended to all types and functions belonging to the package. For example, `struct mc78_control_i`.

This package has two possible usage modes. The first is provided by the subroutines

- `mc78_analyse_asm` and `mc78_analyse_elt` that may be called to perform the full analyse phase. They output information for a factorization phase.

The second mode exposes individual components of the analyse phase and may be used to perform partial analysis of a matrix. The following subroutines are available to the user:

- `mc78_supervars` finds supervariables (columns of A with the same sparsity pattern) and modifies the elimination order so that all variables of a supervariable are consecutive. This modification will not increase the number of entries in L.
- `mc78_compress_by_svar` takes an assembled matrix and its list of supervariables and creates a compressed version of the matrix.
- `mc78_etree` determines the elimination tree of an assembled matrix for a given elimination order.
- `mc78_elt_equiv_etree` finds supervariables of an elemental matrix A, produces an *equivalent matrix* that has the same non-zero pattern in its Cholesky factor as A, and determines the elimination tree of A. Only the lower triangle of the equivalent matrix is returned.
- `mc78_postorder` postorders an elimination tree.
- `mc78_col_counts` computes column counts for the Cholesky factor L, given an assembled matrix and its postordered elimination tree.
- `mc78_supernodes` identifies (relaxed) supernodes of the Cholesky factor L from the elimination tree and column counts.
- `mc78_stats` determines various statistics about the Cholesky factor L given either an elimination or assembly tree and the associated column counts.
- `mc78_row_lists` finds the row indices associated with each supernode of the Cholesky factor L. These may optionally be sorted.
- `mc78_optimize_locality` reorders variables within each supernode in a fashion that attempts to improve cache locality in a later factorization phase.

Both usage modes may need to call

- `mc78_default_control` that sets default values for members of the `mc78_control` data type needed by some other subroutines.

2.2 The derived data types

For some subroutines, the user must employ the structure define in the header file to declare a scalar of type mc78_control. The following pseudocode illustrates this.

```
#include "hsl_mc78i.h"
...
struct mc78_control control;
```

The members of mc78_control are described in Section 2.4.7.

2.3 Argument lists and calling sequences

2.3.1 Package types

We use the following type definitions in this description of the package:

Integer version

```
typedef int pkgtype
```

Long version

```
typedef int64_t pkgtype
```

These types are only used to control whether the input matrix uses 32-bit or 64-bit integers to store column or element pointers. The output representations always use 64-bit integers where they may be required.

2.3.2 The default setting subroutine

Default values for members of the mc78_control structure may be set by a call to mc78_default_control.

```
void mc78_default_control(struct mc78_control *control)
```

control has its members set to their default values, as described in Section 2.4.7

2.3.3 To analyse a matrix

To analyse a matrix, including determining an assembly tree, amalgamating supernodes and finding row lists, the user may call the following routine

For an assembled matrix:

```
int mc78_analyse_asm(int n, const pkgtype ptr[], const int row[],
   int perm[],                int *nnodes, int **sptr, int **sparent, int64_t **rptr,
   int **rlist, const struct mc78_control *control, int *stat, int64_t *nfact, int64_t *nflops,
   int *piv_size);
```

For a matrix in elemental form:

```
int mc78_analyse_elt(int n, int nelt, const pkgtype starts[], const int vars[],
   int perm[], int eparent[], int *nnodes, int **sptr, int **sparent, int64_t **rptr,
   int **rlist, const struct mc78_control *control, int *stat, int64_t *nfact, int64_t *nflops,
   int *piv_size)
```

Unless control.f_arrays evaluates to true, **these routines will use 0-based (C) indexing** for both their inputs and outputs. In the following description of these two routines (only), we assume 0-based indexing of the matrix in the descriptions.

n holds the matrix order (in the element case, this is equal to the largest integer used to index a variable).

ptr is a rank-one array of size n+1. For each column i of A, ptr[i] must specify the position of the first row index of that column in row[], and ptr[n] must specify the total number of entries.

row is a rank-one array of size ptr[n]. The row indices associated with column i of A must be in row[j], j = ptr[i],ptr[i]+1,...,ptr[i+1]-1. Entries in both the lower and upper triangles must be supplied.

nelt holds the number of elements.

starts is a rank-one array of size nelt+1. For each element, starts[elt] must specify the position of the first variable of element elt in vars[], and starts[nelt] must specify the total number of entries in all elements.

vars is a rank-one array of size starts[nelt]. The variables of element elt must be in vars[i], i = starts[elt], starts[elt]+1,...,starts[elt+1]-1.

perm is a rank-one array of size n. On entry, it must be set to hold a permutation that specifies the elimination order. Variable i is pivoted on in position perm[i]. On exit, it specifies a potentially modified elimination order to which the output factor data corresponds. Any modification reflects relabelling of the assembly tree.

eparent is a rank-one array of size nelt. On exit, eparent[i] specifies the supernode corresponding to the least pivot of element i. If element i is empty, eparent[i] will have a value greater than or equal to nnodes.

nnodes specifies, on exit, the number of supernodes.

*sptr will be set, on exit, to hold a pointer to a rank-one array of size nnodes+1 allocated through a call to malloc. It is the user's responsibility to ensure free() is called to release this memory. This array specifies the variables belonging to each supernode. Supernode i contains pivotal variables (*sptr)[i],...,(*sptr)[i+1]-1. If the matrix is rank deficient, not all variables will be part of a supernode: pivots (*sptr)[nnodes],...,n-1 correspond to unused variables (i.e. empty columns).

*sparent will be set, on exit, to hold a pointer to a rank-one array of size nnodes allocated through a call to malloc. It is the user's responsibility to ensure free() is called to release this memory. This array specifies the assembly tree. (*sparent)[i] is the parent of supernode i in the assembly tree. If supernode i is a root then (*sparent)[i] is set to nnodes.

*rptr will be set, on exit, to hold a pointer to a rank-one array of size nnodes+1 allocated through a call to malloc. It is the user's responsibility to ensure free() is called to release this memory. This array specifies the position of first row index of each supernode in *rlist.

*rlist will be set, on exit, to hold a pointer to a rank-one array of size (*rptr)[nnodes] allocated through a call to malloc. It is the user's responsibility to ensure free() is called to release this memory. This array specifies the row lists for each supernode. The indices associated with supernode i are given by (*rlist)[i], i = (*rptr)[i], (*rptr)[i]+1,..., (*rptr)[i+1]-1. The row list indices refer to the elimination order rather than the original (matrix) order. If control.sort evaluates to true, entries within each list are in ascending order.

control is used to control the behaviour of the algorithm, as described in Section 2.4.7.

stat may be NULL. If it is not NULL, on exit *stat will be set to the Fortran stat value from the last call made to allocate or deallocate.

nfact may be NULL. If it is not NULL, *nfact will be set on exit to the number of entries in the Cholesky factor L with the given supernode pattern.

nflops may be NULL. If it is not NULL, *nflops will be set on exit to the number of floating-point operations required to calculate the Cholesky factor L with the given supernode pattern.

piv_size may be NULL. If it is not NULL, it must point to a rank-one array of size n. On entry this array specifies block pivots to be used and on exit it specifies the same information modified to match any changes to perm. The value piv_size[i] gives the number of pivots in the block pivot containing variable i of A. If a block pivot contains an unused variable, then that variable will be removed from the block pivot and placed at the end of the elimination order. Note that in the current version of HSL_MC78, the use of block pivots can significantly increase the time and memory required for analysis.

Return code

On normal completion the return code will be zero: other values indicate a warning or error return as described in Section 2.5.

2.3.4 To identify supervariables of an assembled matrix

To identify supervariables of an assembled matrix, the user may call the routine

```
int mc78_supervars(int *n, const pkgtype ptr[], const int row[], int perm[], int invp[],
   int *nsvar, int svar[])
```

*n holds the matrix order. On exit it specifies the number of variables that are actually used (i.e. non-empty columns).

ptr is a rank-one array of size n+1. For each column i of A, ptr[i-1]-1 must specify the position of the first row index of that column in row[], and ptr[n] must specify the total number of entries plus one.

row is a rank-one array of size ptr[n]-1. The row indices associated with column i of A are given by row[j], j = ptr[i-1]-1,...,ptr[i]-2. Entries in both the lower and upper triangles must be supplied.

perm and invp are rank-one arrays of size n. On entry they describe a permutation and its inverse such that if row i is the j-th pivot, then perm[i-1]=j, and invp[j-1]=i. On exit, the permutation is rearranged such that all variables in a given supervariable are consecutive (and take a position in the elimination order equivalent to the first variable of the supervariable). Any unused variables (i.e. empty columns) are permuted to the end of the elimination order.

*nsvar is set, on exit, to the number of supervariables identified.

svar is a rank-one array of size n. On exit, the first nsvar entries specify the number of variables in each supervariable.

Return code

On normal completion the return code will be zero: other values indicate a failed allocation, and specify the value of the Fortran stat parameter returned by the failed call.

2.3.5 To compress an assembled matrix using supervariables

To compress an assembled matrix using previously identified supervariables, to obtain the symmetric matrix (in full storage) with columns and rows corresponding to supervariables in elimination order, the user may call the routine

```
int mc78_compress_by_svar(int n, const pkgtype ptr[],const int row[], const int invp[],
   int nsvar, const int svar[], pkgtype ptr2[], pkgtype lrow2, int row2[], int *st);
```

`n` holds the matrix order.

`ptr` is a rank-one array of size `n+1`. For each column `i` of A, `ptr[i-1]-1` must specify the position of the first row index of that column in `row[]`, and `ptr[n]` must specify the total number of entries plus one.

`row` is a rank-one array of size `ptr[n]-1`. The row indices associated with column `i` of A are given by `row[j]`, `j` = `ptr[i-1]-1`,...,`ptr[i]-2`. Entries in both the lower and upper triangles must be supplied.

`invp` is a rank-one array of size `n`. It describes an inverse permutation such that if row `i` is the `j`-th pivot, then `invp[j-1]=i`.

`nsvar` must hold the number of supervariables.

`svar` is a rank-one array of size `nsvar`. It must hold the number of variables in each supervariable.

`ptr2` is a rank-one array of size `n+1`. For each column `i` of the compressed matrix, `ptr2[i-1]-1` specifies the first row index of that column in `row2[]`, and `ptr2[n]` specifies the total number of entries plus one.

`lrow2` specifies the length of the array `row2`. It should be sufficiently large that `row2` can store the row indices of the compressed matrix. An upper limit on the required size is `ptr[n]-1`.

`row2` is a rank-one array of size `lrow2`. On exit, the row indices associated with column `i` of the compressed matrix are given by `row2[i]`, `i` = `ptr2[i-1]-1`,...,`ptr2[i]-2`. Columns are specified in elimination order and entries in both the lower and upper triangle are stored. If the size of `row2` is insufficient, an error is returned.

`*st` contains, on exit, the Fortran stat value of the last allocate call executed by the subroutine. It will only be non-zero in the case of an error.

Return code

On normal completion the return code will be zero: other values indicate a warning or error return as described in Section 2.5.

2.3.6 To determine the elimination tree of an assembled matrix

To determine the elimination tree of a matrix under a given elimination order, the user may call the routine

```
int mc78_etree(int n, const pkgtype ptr[], const int row[], const int perm[],
   const int invp[], int parent[])
```

`n` specifies the number of rows and columns in the matrix.

`ptr` is a rank-one array of size `n+1`. For each column `i` of A, `ptr[i-1]-1` must specify the position of the first row index of that column in `row[]`, and `ptr[n]` must specify the total number of entries plus one.

`row` is a rank-one array of size `ptr[n]-1`. The row indices associated with column `i` of A are given by `row[i]`, `i` = `ptr[i-1]`,...,`ptr[i]-2`. Entries in both the lower and upper triangles must be supplied.

`perm` and `invp` are rank-one arrays of size `n`. They describe a permutation and its inverse such that if row `i` is the `j`-th pivot, then `perm[i-1]=j`, and `invp[j-1]=i`.

`parent` is a rank-one array of size `n`. On exit, `parent[i-1]` specifies the parent of node `i` in the elimination tree, or has the value `n+1` if node `i` is a root.

Return code

On normal completion the return code will be zero: other values indicate a failed allocation, and specify the value of the Fortran stat parameter returned by the failed call.

2.4 To find supervariables, equivalent matrix and elimination tree of an element problem

Supervariables of an *elemental matrix* are determined and the lower triangle of an assembled symmetric matrix is returned. The Cholesky factor of this assembled matrix under natural ordering will have the same pattern as the elemental matrix under the supplied ordering, but it is compressed using supervariables to use less space. As the lower triangular form is unsuitable for determining the elimination tree, the elimination tree is determined for the user during this construction.

```
int mc78_elt_equiv_etree(int *n, int nelt, const pkgtype starts[], const int vars[],
   int perm[], int invp[], int *nsvar, int svar[], pkgtype ptr[], int row[],
   int eparent[], int parent[], int block_pivots[])
```

`*n` specifies the matrix order on entry. On exit it specifies the number of variables that are actually used.

`nelt` specifies the number of elements.

`starts` is a rank-one array of size `nelt+1`. For each element, `starts[elt-1]-1` must specify the position of the first variable of element `elt` in `vars[]`, and `starts[elt]-2` must specify the last.

`vars` is a rank-one array of size `starts[nelt]-1`. The variables of element `elt` are given by `vars[i]`, `i` = `starts[elt-1]-1`,...,`starts[elt]-2`.

`perm` and `invp` are rank-one arrays of size `n`. On entry they describe a permutation and its inverse such that if row `i` is the `j`-th pivot, then `perm[i-1]=j`, and `invp[j-1]=i`. On exit, the permutation is rearranged such that all variables in a given supervariable are consecutive (and take a position in the elimination order equivalent to the first variable of the supervariable). Any unused variables are permuted to the end of the elimination order.

`*nsvar` contains, on exit, the number of supervariables in the compressed form.

`svar` is a rank-one array of size `n`. On exit, the first `*nsvar` entries will have been set such that `svar[i-1]` gives the number of variables in supervariable `i`. The supervariables are numbered by the order they appear in the input permutation. Thus the first `svar[0]` entries of `invp` give the variables in supervariable 1, the next `svar[1]` entries the variables of supervariable 2 and so forth.

`ptr` is a rank-one array of size `n+1`. On output it specifies the column pointers of a lower triangular assembled matrix whose Cholesky factor has the same non-zero pattern as the elemental input matrix. The matrix is compressed through the use of supervariables. For each supercolumn `i` of the matrix, `ptr[i]` specifies the position of the first superrow index of supercolumn *i* in `row[]`, and `ptr[i]-2` specifies the last.

`row` is a rank-one array of size `starts[nelt]-1`. On output it specifies the row indices of a lower triangular assembled matrix whose Cholesky factor has the same non-zero pattern as the elemental input matrix. The matrix is compressed through the use of supervariables. The superrow indices associated with supercolumn `i` of *A* are given by `row[j]`, `j` = `ptr[i-1]`,...,`ptr[i]-2`. Entries in only the lower triangle and not on the diagonal are supplied.

`eparent` is a rank-one array of size `nelt`. On exit, `eparent[i-1]` specifies the variable corresponding to the least pivot of element *i*. If element `i` is empty, `eparent[i-1]` will have a value greater than `n`.

parent is a rank-one array of size nsvar. On exit, parent[i-1] specifies the parent of node i in the elimination tree, or has the value nsvar+1 if node i is a root.

block_pivots may be NULL. If it is not NULL, it must be a pointer to a rank-one array of size n. This array indicates block pivots that will be amalgamated into the same supernode. block_pivots[i-1] corresponds to pivot i, and takes one of the following values:

0 Pivot i is neither the first nor the last pivot of a block pivot.

1 Pivot i is the first pivot of a block pivot.

2 Pivot i is the last pivot of a block pivot.

3 Pivot i is a 1×1 pivot.

On exit, the array will be modified to reflect any changes to perm. If a block pivot contains unused variables, then they are removed and placed at the end of the elimination order.

Return code

On normal completion the return code will be zero: other values indicate a failed allocation, and specify the value of the Fortran stat parameter returned by the failed call.

2.4.1 To postorder an elimination tree

Given a matrix, elimination order and the corresponding elimination tree the user may call the following routine to modify the elimination order such that the elimination tree becomes postordered. That is to say, the numbering of each node and its descendants is such that for each node b, there exists a minimal descendant a, and the set of all descendants of b is the sequence $a, a+1, \ldots b-1$.

```
int mc78_postorder(int n, int *realn, const pkgtype ptr[], int perm[], int invp[],
   int parent[], int block_pivots[])
```

If realn and ptr are not NULL then additional work is performed to ensure any pivots corresponding to unused variables (i.e. empty columns) are moved to the end of the elimination order.

n specifies the order of the matrix.

realn may be NULL. If it is NULL, then ptr must also be NULL. If it is not NULL, *realn will, on exit, hold the number of variables that are actually used.

ptr may be NULL. If it is NULL, then realn must also be NULL. If it is not NULL, it must point to a rank-one array of size n+1. The number of entries in column i of A must be equal to ptr[i]-ptr[i-1].

perm and invp are rank-one arrays of size n. On entry they describe a permutation and its inverse such that if row i is the j-th pivot, then perm[i-1]=j, and invp[j-1]=i. On exit they are modified such that the elimination tree specified by parent is postordered.

parent is a rank-one array of size n. On entry, parent[i-1] specifies the parent of node i in the elimination tree, or has the value n+1 if node i is a root. On exit, it is modified to match the new ordering given by perm.

block_pivots may be NULL. If it is not NULL it must point to a rank-one array of size n. This array will be modified to match the new ordering given by invp.

Return code

On normal completion the return code will be zero: other values indicate a failed allocation (specifying the value of the Fortran stat parameter returned by the failed call), or exactly one of ptr or realn was NULL (value -1).

2.4.2 To determine column counts of a Cholesky factor

To determine the number of entries in each column of the Cholesky factor L of a matrix A given a postordering and associated elimination tree, the user may call the routine

```
int mc78_col_counts(int n, const pkgtype ptr[], const int row[], const int perm[],
   const int invp[], const int parent[], int cc[], const int wt[])
```

`n`, `ptr` and `row` are as described in Section 2.3.6.

`perm` and `invp` are rank-one arrays of size `n`. They describe a permutation and its inverse such that if row `i` is the `j`-th pivot, then `perm[i-1]=j`, and `invp[j-1]=i`.

`parent` is a rank-one array of size `n`. It must describe a postordered elimination tree corresponding to the matrix described by `n`, `ptr` and `row` under the elimination order given by `perm`. The entry `parent[i-1]` must specify the parent of node `i`, or have the value `n+1` if node `i` is a root.

`cc` is a rank-one array of size `n+1`. On exit, `cc[i-1]` gives the number of entries in column `i` of the Cholesky factor of PAP^{-1}.

`wt` may be `NULL`. If it is not `NULL`, it must point to a rank-one array of size `n`. In this case, column and row `i` of the matrix is treated as if it were in fact `wt[i-1]` rows or columns with the same sparsity pattern; the column counts output in `cc` will correspond to the number of rows in the uncompressed matrix.

Return code
On normal completion the return code will be zero: other values indicate a failed allocation, and specify the value of the Fortran stat parameter returned by the failed call.

2.4.3 To identify supernodes

To identify (relaxed) supernodes of a Cholesky factor L given by an elimination tree and associated column counts, the user may call the routine

```
int mc78_supernodes(int n, int realn, const int parent[], const int cc[], int sperm[],
   int *nnodes, int sptr[], int sparent[], int scc[], const int invp[],
   const struct mc78_control *control, int *st, const int wt[], const int block_pivots[])
```

`n` specifies the order of the matrix.

`realn` specifies the number variables that are used in the description of A (empty columns need not be included in any supernode).

`parent` is a rank-one array of size `n`. It describes a postordered elimination tree, such that `parent[i-1]` specifies the parent of node `i`, or has the value `n+1` if node `i` is a root.

`cc` is a rank-one array of size `n`. The number of entries in column `i` of L is `cc[i-1]`.

`sperm` is a rank-one array of size `n`. On exit, it contains a map from the elimination order to a new order where supernodes are contiguous. If `sperm[i-1]=j`, then pivot i is now pivot j.

`*nnodes` contains, on exit, the number of supernodes found.

`sptr` is a rank-one array of size `n+1`. On exit, the first `nnodes+1` elements will be set such that supernode `i` will consist of pivots `sptr[i-1]` through `sptr[i]-1`.

sparent is a rank-one array of size n. On exit, the first nnodes will describe the assembly tree such that sparent[i-1] specifies the parent of supernode i, or nnodes+1 if supernode i is a root.

scc is a rank-one array of size n. On exit, the first nnodes entries will be set such that scc[i-1] gives the number of rows in supernode i of L.

invp is a rank-one array of size n. It describes an inverse permutation such that if row i is the j-th pivot, then invp[j-1]=i.

control controls the supernode amalgamation heuristics used by the routine, as described in Section 2.4.7.

*st contains, on exit, the stat value of the last allocate call executed by the subroutine. This will only be non-zero in the case of an allocation error.

wt may be NULL. If it is not NULL it must point to a rank-one array of size n. In this case, column i of the matrix described by parent and cc is treated as if it were in fact wt[i-1] columns with the same sparsity pattern for the purposes of supernode amalgamation heuristics.

block_pivots may be NULL. If it is not NULL it must point to a rank-one array of size n. In this case, it indicates block pivots that will be amalgamated into the same supernode. block_pivots[i-1] corresponds to pivot i, and takes one of the following values:

- 0 Pivot i is neither the first nor the last pivot of a block pivot.
- 1 Pivot i is the first pivot of a block pivot.
- 2 Pivot i is the last pivot of a block pivot.
- 3 Pivot i is a 1×1 pivot.

Return code

On normal completion the return code will be zero: other values indicate a warning or error return as described in Section 2.5.

2.4.4 To determine statistics about a Cholesky factor

To determine the number of entries in the Cholesky factor L and number of floating-point operations required to compute L, given the supernode distribution and associated row counts, the user may call the routine

```
void mc78_stats(int nnodes, const int sptr[], const int scc[], int64_t *nfact, int64_t *nflops)
```

nnodes specifies the number of supernodes.

sptr is a rank-one array of size nnodes+1. It must be set so that supernode i consists of pivots sptr[i-1] through sptr[i]-1.

scc is a rank-one array of size nnodes. It must be set so that supernode i of L has scc[i-1] rows.

nfact may be NULL. If it is not NULL, *nfact will be set on exit to the number of entries in the Cholesky factor L with the given supernode pattern.

nflops may be NULL. If it is not NULL, *nflops will be set on exit to the number of floating-point operations required to calculate the Cholesky factor L with the given supernode pattern.

Return code

On normal completion the return code will be zero: other values indicate a warning or error return as described in Section 2.5.

2.4.5 To determine the row indices of a supernodal Cholesky factor

To determine the row lists for each supernode of a Cholesky factor L of the matrix A, the user may call the subroutine

```
int mc78_row_lists(int nsvar, const int svar[], int n, const pkgtype ptr[],
   const int row[], const int perm[], const int invp[], const int nnodes,
   const int sptr[], const int sparent[], const int scc[], int64_t rptr[], int rlist[],
   const struct mc78_control *control, int *st)
```

For non-supervariable representations, `svar` should be set to `NULL`.

`nsvar` specifies the number of supervariables used to compress the matrix.

`svar` is a rank-one array argument of size `nsvar`. It should be set such that supervariable `i` contains `svar[i-1]` variables.

`n`, `ptr` and `row` are as described in Section 2.3.6.

`perm` and `invp` are rank-one arrays of size `n`. They describe a permutation and its inverse such that if row `i` is the `j`-th pivot, then `perm[i-1]=j`, and `invp[j-1]=i`.

`nnodes` specifies the number of supernodes.

`sptr` is a rank-one array of size `nnodes+1`. It must be set so that supernode `i` consists of pivots `sptr[i-1]` through `sptr[i]-1`.

`sparent` is a rank-one array of size `nnodes`. It describes the assembly tree, such that `sparent[i-1]` specifies the parent of supernode `i`, or has the value `nnodes+1` if supernode `i` is a root.

`scc` is a rank-one array of size `nnodes`. It must be set so that supernode `i` of L has `scc[i-1]` rows.

`rptr` is a rank-one array of size `nnodes+1`. On exit, it specifies the positions in `rlist` that contain the row lists for each supernode.

`rlist` is a rank-one array of size equal to the sum of `scc[i]`, `i` $= 0,\ldots,$ `nnodes-1`. On exit, the row list for supernode `i` is given by the entries `rlist[j]`, `j` $=$ `rptr[i-1]-1`,$\ldots$,`rptr[i]-2`.

`control` controls printing of error information from the routine, as described in Section 2.4.7.

`*st` is as described in Section 2.3.6.

Return code

On normal completion the return code will be zero: other values indicate a warning or error return as described in Section 2.5.

2.4.6 To optimize variable ordering for cache locality

To permute variables within supernodes to improve cache locality in sparse update operations in a subsequent factorization phase, the user may call the routine

```
int mc78_optimize_locality(int n, int realn, int perm[], int invp[], const int nnodes,
   const int sptr[], const int sparent[], const int64_t rptr[], int rlist[], int sort)
```

`n`, `nnodes`, `sptr`, and `sparent` are as described in Section 2.4.5.

realn specifies the number of variables that are actually used in A (i.e. the number of non-empty columns).

perm and invp are rank-one arrays of size n. On entry they describe a permutation and its inverse such that if row i is the j-th pivot, then perm[i-1]=j, and invp[j-1]=i. On exit, they have been modified to give a new ordering with the same supernodes but better cache locality in the factorization phase.

rptr is a rank-one array of size nnodes+1. It specifies the positions in rlist that contain the row lists for each supernode.

rlist is a rank-one array of size equal to the sum of scc[i], i = 0,...,nnodes-1. On entry, the row list for supernode i is given by the entries rlist[j], j = rptr[i-1]-1,...,rptr[i]-2. On exit, each row list has been updated to match the new permutation.

sort controls sorting. If it evaluates to true, the entries of each row list are returned in ascending order. Otherwise, if it evaluates to false, the entries of each row list may be in any order.

Return code

On normal completion the return code will be zero: other values indicate a failed allocation, and specify the value of the Fortran stat parameter returned by the failed call.

2.4.7 The derived data type for holding control parameters

The derived data type mc78_control is used to hold controlling data. The components, which are automatically given default values upon instantiation, are:

int nemin controls the node amalgamation heuristic. A node and its parent are merged if both have fewer than nemin columns. It has default value 16.

int lopt controls cache locality optimizations. If lopt evaluates to true, mc78_analyse_asm and mc78_analyse_elt will reorder variables within each supernode to attempt to maximize cache locality in the factorize phase. If, in the call to mc78_analyse_asm, the optional argument piv_size is present, pivots within blocks may be separated, but will remain within the same supernode. The default value is 0 (false).

int sort controls sorting of variable lists. If sort evaluates to true, then on return from a call to mc78_analyse, the entries of each supernode row list are sorted into ascending order. Otherwise, these values are returned in arbitrary order. The default value is 0 (false).

int ssa_abort controls the action when an assembled matrix is found to be symbolically singular (a row or column contains no entries) during a call to mc78_analyse. If ssa_abort evaluates to true an error is raised as soon as rank deficiency is detected. If ssa_abort evaluates to false a warning is raised, but computation then proceeds as normal, but with empty columns moved to the end of the elimination order. The default value is 0 (false).

int svar controls use of supervariables. If svar evaluates to true, mc78_analyse_asm will identify and exploit supervariables (mc78_analyse_elt always uses supervariables). Otherwise supervariables are not exploited. Note that if, in the call to mc78_analyse_asm, the argument piv_sizes is not NULL, then supervariables are not exploited regardless of the value of this control. The default value is 0 (false).

int unit_error controls the printing of error messages. If positive, then errors are printed on the Fortran unit unit_error. Otherwise error messages are suppressed. The default value is 6.

int unit_warning controls the printing of warning messages. If positive, then warnings are printed on the Fortran unit unit_warning. Otherwise warning messages are suppressed. The default value is 6.

2.5 Warning and error messages

A successful return is indicated by a return code of zero. Unless otherwise indicated (such as when the return code is a Fortran stat value), a negative value is associated with an error, and a positive value is associated with a warning. Possible negative values are:

`-1` A memory allocation error has occurred.

`-2` Matrix is symbolically singular, assembled and `control.ssa_abort` evaluates to true.

`-3` The array `row` has insufficient size to store the new matrix.

Possible positive values are:

`+1` Matrix is symbolically singular, assembled and `control.ssa_abort` evaluates to false.

`+2` Both supervariables and block pivots have been requested. These options are not compatible so only block pivots were used.

`+3` Both warnings `+1` and `+2`.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: None.

Input/output: Warnings and errors are printed on Fortran units `control.unit_warning` and `control.unit_error` respectively. If the supplied units are negative, output is suppressed.

Portability: Fortran 2008 subset (F95 + TR15581 + C interoperability).

4 METHOD

What follows is broad overview of the algorithms used in this package. A more detailed description, including numerical results, is included in the following report:

J.D. Hogg and J.A. Scott. *A modern analyse phase for sparse tree-based direct methods.* RAL-TR-2010-031.

4.1 Analyse Method

`mc78_analyse` calls other subroutines in order to determine most of its information.

In the assembled case, if supervariables are requested, they are identified and the matrix is compressed. In the element case, supervariables are determined and a compressed equivalent matrix is built. In both cases, the elimination tree is then determined. The assembled case calls `mc78_supervars`, `mc78_compress_by_svar` and `mc78_etree` to do this. The elemental case combines its operations for efficiency in a single call to `mc78_elt_equiv_etree`.

The elimination tree is postordered using `mc78_postorder`, allowing a call to `mc78_col_counts` to find the column counts. This gives sufficient information to perform supernode amalgamation with `mc78_supernodes`. All that then remains is to call `mc78_row_lists` to determine the supernodal sparsity pattern.

Finally, information is expanded from the compressed form if necessary and statistics are calculated through a call to `mc78_stats` if requested. If `control.lopt` evaluates to true then cache locality optimizations are performed by `mc78_optimize_locality`. If `control.sort` evaluates to true, row lists are sorted using a double transpose sort.

No checking of data validity is performed at any stage.

4.2 Identifying supervariables

The identification of supervariables is done with an algorithm based on that described in [1]. It has been modified to reduce the amount of data movement involved with supervariables that contain only a single variable.

4.3 Determining the elimination tree

The elimination tree is determined through the use of an algorithm due to Liu [2]. This needs only to access the entries in the upper half of the matrix (once in elimination order) and entries in the lower part are ignored.

4.4 The combined call `mc78_elt_equiv_etree`

In the element case, the processes of identifying supervariables and building the equivalent matrix are combined. In fact two equivalent matrices are built: one lower and one upper triangular, both preordered so that the elimination order is $1,2,3,4,\ldots n$. The upper triangular matrix is used to find the elimination tree only, and we exploit the fact that we do not need to access the permutation in this case. The lower triangular matrix is returned for use in further analysis.

4.5 Postordering the elimination tree

The elimination tree is postordered through a simple depth-first search. It is designed such that the relative ordering of the children of each node is preserved.

4.6 Finding column counts

Column counts are found in time proportional to the number of non-zeros in A using the algorithm of Gilbert, Ng and Peyton [3]. We have specialised their original algorithm to find only column counts (the original found row counts also). The algorithm has been modified to allow for optionally weighting the columns in order to support supervariables.

4.7 Supernode amalgamation

We loop over the nodes of the elimination tree in depth first order. If either of the following conditions hold in the current assembly tree, a node is merged into its parent:

- Merging the parent and child will not introduce additional non-zeros in the factors.
- The parent and child each have fewer than `nemin` columns.

4.8 Determining statistics

The number of entries in L and the number of floating-point operations to perform a Cholesky factorization may be calculated given the supernode partition and column counts associated with each supernode. From this we can determine for the column count for each column of L. If we denote this count for column i as $cc(i)$, then the number of entries in the factors is given by

$$nfact = \sum_{i=1}^{n} cc(i),$$

and the number of floating point operations is given by

$$nflops = \sum_{i=1}^{n} cc(i)^2.$$

4.9 Finding the row lists

Given a supernode partition and associated assembly tree we can perform a straightforward symbolic factorization to identify the row lists. With the supernodal column counts available we can avoid expensive data reorganisations. Working in a depth-first order, the non-zero set of a supernode is obtained by merging the non-zero sets of its children in the assembly tree and the non-zero sets of its associated columns in the original matrix A.

4.10 Optimizing for cache locality

Given a particular supernode partition (that is assignment of variables to supernodes), there is still a freedom in ordering the variables within a supernode. The subroutine `mc78_optimize_locality` exploits this freedom to increase the cache locality in a future numerical factorization phase by maximizing the number of entries that are are used contiguously in sparse expansion operations.

The algorithm used to do this is as follows. A depth-first search order is established on the tree such that of the children at each node are ordered by the number of variables they pass to the parent, with the largest first. Loop over supernodes in this order, and examine variables present. If a variable is being encountered for the first time, order it in the first available position corresponding to its supernode.

4.11 References

[1] I.S. Duff and J.K. Reid. 1996. *Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems.* ACM TOMS 22, 2. pp227-257.

[2] J.W. Liu, 1986. *A compact row storage scheme for Cholesky factors using elimination trees.* ACM TOMS 12, 2. pp127–148.

[3] Gilbert, Ng, Peyton. 1994. *An efficient algorithm to compute row and column counts for sparse Cholesky factorization.* SIMAX 15, 4.

[4] J.D. Hogg and J.A. Scott, 2010. *A modern analyse phase for sparse tree-based direct methods.* RAL-TR-2010-031. Available from `http://www.stfc.ac.uk/CSE/36276.aspx`.

5 EXAMPLE OF USE

5.1 Assembled matrix example

To analyse an assembled matrix and obtain the size of the factors and number of floating point operations the user may use the following code

```
/* hsl_mc78is.c */
#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc78i.h"

typedef int pkgtype;

int main(void) {
   int i, j, n, nnodes, info;
   int *row, *perm, *sptr, *sparent, *rlist;
   pkgtype *ptr;
```

```
   int64_t *rptr, nfact, nflops;
   struct mc78_control control;

   /* Read matrix from standard input */
   scanf("%d\n", &n); /* Matrix dimension */
   ptr = (pkgtype *) malloc((n+1)*sizeof(pkgtype));
   perm = (int *) malloc(n*sizeof(int));
   for(i=0; i<n+1; i++) scanf("%d", &ptr[i]); /* Column pointers */
   row = (int *) malloc(ptr[n]*sizeof(int));
   for(i=0; i<ptr[n]; i++) scanf("%d", &row[i]); /* Row indices */

   /* Use identity as pivot order */
   for(i=0; i<n; i++) perm[i] = i;

   /* Perform analysis */
   /* (mallocs sptr, sparent, rptr and rlist: these need explicitly freed) */
   mc78_default_control(&control); /* initialise to defaults */
   control.nemin = 1; /* Disable supernode amalgamation as such small matrix */
   info = mc78_analyse_asm(n, ptr, row, perm, &nnodes, &sptr, &sparent, &rptr,
         &rlist, &control, NULL, &nfact, &nflops, NULL);
   if(info < 0) {
      printf("mc78_analyse returned with unexpected error %d\n", info);
      return 1;
   }

   /* Print out results */
   for(i=0; i<nnodes; i++) {
      printf("Node %d (columns %d to %d)\n", i, sptr[i], sptr[i+1]-1);
      printf("   parent in assembly tree is %d\n", sparent[i]);
      printf("   row list is:\n      ");
      for(j=rptr[i]; j<rptr[i+1]; j++) printf("%d ", rlist[j]);
      printf("\n");
   }
   printf("\nTotal number of entries in factor = %ld\n", nfact);
   printf("Total number of floating point operations = %ld\n", nflops);

   /* Perform a clean exit */
   free(ptr); free(row); free(perm);                    /* allocated by us */
   free(sptr); free(sparent); free(rptr); free(rlist); /* allocated by mc78 */
   return 0;
}
```

For the matrix

$$\begin{pmatrix} x & & x & & \\ & x & x & & x \\ x & x & x & & \\ & & & x & x \\ & x & & x & x \end{pmatrix},$$

the following input

```
 5
 0       2       5       8      10
13
 0       2       1       2       4
 0       1       2       3       4
 1       3       4
```

is suitable and yields the following output

```
Node 0 (columns 0 to 0)
   parent in assembly tree is 2
   row list is:
      0 3
Node 1 (columns 1 to 1)
   parent in assembly tree is 2
   row list is:
      1 4
Node 2 (columns 2 to 4)
   parent in assembly tree is 3
   row list is:
      2 3 4

Total number of entries in factor = 10
Total number of floating point operations = 22
```

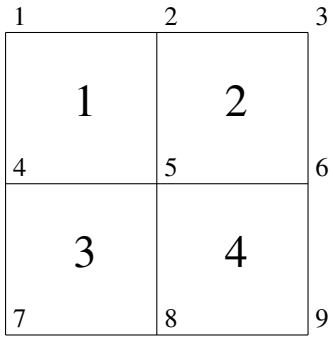
5.2 Element case example

To analyse an elemental matrix with a block pivot $(4,5)$ the user may use the following code

```
/* hsl_mc78is1.c */
#include <stdio.h>
#include <stdlib.h>
#include "hsl_mc78i.h"

typedef int pkgtype;

int main(void) {
   int i, j, n, nelt, nnodes, info;
   int *vars, *perm, *eparent, *sptr, *sparent, *rlist, *piv_size;
   pkgtype *starts;
   int64_t *rptr;
   struct mc78_control control;

   /* Read matrix from standard input */
   scanf("%d %d\n", &n, &nelt); /* Matrix dimension and number of elements */
   starts = (pkgtype *) malloc((nelt+1)*sizeof(pkgtype));
   perm = (int *) malloc(n*sizeof(int));
   eparent = (int *) malloc(nelt*sizeof(int));
   piv_size = (int *) malloc(n*sizeof(int));
   for(i=0; i<nelt+1; i++) scanf("%d", &starts[i]); /* Element pointers */
   vars = (int *) malloc((starts[nelt])*sizeof(int));
   for(i=0; i<starts[nelt]; i++) scanf("%d", &vars[i]); /* Element variables */

   /* Use identity as pivot order and define (4,5) as a block pivot */
   for(i=0; i<n; i++) perm[i] = i;
   for(i=0; i<n; i++) piv_size[i] = 1;
   piv_size[3] = 2; piv_size[4] = 2;

   /* Perform analysis */
   mc78_default_control(&control);
   control.nemin = 1; /* Disable supernode amalgamation as such small matrix */
   info = mc78_analyse_elt(n, nelt, starts, vars, perm, eparent, &nnodes,
         &sptr, &sparent, &rptr, &rlist, &control, NULL, NULL, NULL, piv_size);
   if(info < 0) {
      printf("mc78_analyse returned with unexpected error %d\n", info);
      return 1;
   }

   /* Print out results */
   for(i=0; i<nnodes; i++) {
      printf("Node %d (columns %d to %d)\n", i, sptr[i], sptr[i+1]-1);
      printf("   parent in assembly tree is %d\n", sparent[i]);
      printf("   row list is:\n      ");
      for(j=rptr[i]; j<rptr[i+1]; j++) printf("%d ", rlist[j]);
      printf("\n");
```

```
   }
   for(i=0; i<nelt; i++) {
      printf("Element %d is a child of node %d\n", i, eparent[i]);
   }

   /* Perform a clean exit */
   free(starts); free(vars); free(perm); free(piv_size); free(eparent); /* us */
   free(sptr); free(sparent); free(rptr); free(rlist); /* allocated by mc78 */
   return 0;
}
```

For the element problem in the following picture,



with variables at nodes 3, 6 and 9 fixed, the following input

```
       8       4
       0       4       6      10      12
       0       1       3       4       1
       4       3       4       6       7
       4       7
```

is suitable and yields the following output

```
Node 0 (columns 0 to 1)
   parent in assembly tree is 1
   row list is:
      0 1 2 3
Node 1 (columns 2 to 5)
   parent in assembly tree is 2
   row list is:
      2 3 4 5
Element 0 is a child of node 0
Element 1 is a child of node 0
Element 2 is a child of node 1
Element 3 is a child of node 1
```