1 SUMMARY

Let K be an $n \times n$ sparse symmetric saddle-point matrix of the form

$$K = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix},$$

where A is $(n-m) \times (n-m)$ symmetric positive definite, B is rectangular $m \times (n-m)$ and of full rank ($m < n$), and C is $m \times m$ symmetric positive semi-definite. `HSL_MI30` **computes a signed incomplete Cholesky factorization**. That is, a factorization of the form LDL^T, where L is lower triangular and D is diagonal with $n-m$ positive entries and m negative entries. The matrix K is optionally reordered, scaled and, if necessary, shifted to avoid breakdown of the factorization so that the LDL^T incomplete factorization of the matrix

$$\bar{K} = SQ^T \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} QS + \begin{pmatrix} \alpha(1)I & 0 \\ & -\alpha(2)I \end{pmatrix}$$

is computed, where Q is a permutation matrix, S is a diagonal scaling matrix and $\alpha(1:2)$ are non-negative shifts.

The incomplete factorization may be used for preconditioning when solving the saddle-point system $Kx = b$. A separate entry performs the preconditioning operation

$$y = Pz$$

where $P = (\bar{L}\,D\,\bar{L}^T)^{-1}$, with $\bar{L} = QS^{-1}L$, is the incomplete signed Cholesky factorization preconditioner. An option exists to use $P = (\bar{L}\,|D|\,\bar{L}^T)^{-1}$ as the preconditioner.

The incomplete factorization is based on a matrix decomposition of the form

$$\overline{K} = (L+R)\,D\,(L+R)^T - E, \tag{1.1}$$

where L is lower triangular with unit diagonal entries, R is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process, D is a diagonal matrix, and E has the form

$$E = RDR^T + F + F^T, \tag{1.2}$$

where F is strictly lower triangle. E is not computed explicitly and all terms in F are ignored, while the matrix R is used in the computation of L but is then discarded. The user controls the dropping of small entries from L and R and the maximum number of entries within each column of L and R (and thus the amount of memory for L and the intermediate work and memory used in computing the incomplete factorization).

Note: If an incomplete Cholesky factorization preconditioner for a symmetric positive-definite system is required, `HSL_MI28` should be used.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a `USE` statement

Single precision version
```
use hsl_mi30_single
```
Double precision version
```
use hsl_mi30_double
```
If it is required to use more than one module at the same time, the derived types (see Section 2.2) must be renamed in one of the `USE` statements.

The following procedures are available to the user:

(a) `mi30_factorize` computes an incomplete signed Cholesky factorization.

(b) `mi30_precondition` performs the preconditioning operation $y = Pz$, where P is the incomplete factorization preconditioner computed by `mi30_factorize`.

(c) `mi30_solve` solves the system $\overline{L}\, D\, y = SQ^T z$ (or $\overline{L}\, |D| y = SQ^T z$ or $\overline{L}^T S^{-1} Q^T y = z$), where $\overline{L}$ is the incomplete factor computed by `mi30_factorize` and $|D|$ has entries $|d_{ij}|$.

(d) `mi30_finalise` frees memory that has been allocated by `mi30_factorize`.

2.2 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types `mi30_keep`, `mi30_control` and `mi30_info`. The following pseudocode illustrates this.

```
use hsl_mi30_double
...
type (mi30_keep) :: keep
type (mi30_control) :: control
type (mi30_info) :: info

...
```

The components of `mi30_control` and `mi30_info` are explained in Sections 2.5 and 2.6. The components of `mi30_keep` are used to pass data between the subroutines of the package and must not be altered by the user.

2.3 METIS

The `HSL_MI30` package optionally uses the METIS graph partitioning library available from the University of Minnesota website. If METIS is not available, the user must link with the supplied dummy subroutine `METIS_NodeND`. In this case, the METIS ordering option will not be available to the user and, if selected, `mi30_factorize` will return with an error.

Important: At present, `HSL_MI30` only supports METIS version 4, not the latest version 5 releases.

2.4 Argument lists and calling sequences

2.4.1 Optional arguments

We use square brackets `[ ]` to indicate `OPTIONAL` arguments. In each call, optional arguments follow the argument `info`. Since we reserve the right 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**.

2.4.2 Integer and package types

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

2.4.3 To compute a signed incomplete Cholesky factorization

To compute a signed incomplete Cholesky factorization, the lower triangular part of the matrix K must be held in compressed column storage and a call of the following form must be made:

```
call mi30_factorize(n, m, ptr, row, val, lsize, rsize, keep, control, info[, scale, perm])
```

n is an `INTENT(IN)` scalar of type `INTEGER` that must hold the order of the matrix K. **Restriction:** $2 \leq$ n.

m is an `INTENT(IN)` scalar of type `INTEGER` that must hold the order of the $(2,2)$ block $-C$. **Restriction:** $1 \leq$ m $\leq$ n-1.

ptr is an `INTENT(INOUT)` rank-1 array of type `INTEGER` and size n+1. `ptr(j)` must be set by the user so that `ptr(j)` is the position in `row` of the first entry in column `j` and `ptr(n+1)` must be set to one more than the number of matrix entries being input by the user. `ptr` is only changed on exit if `control%check` is set to `.true.` (the default) and duplicates and/or out-of-range indices are found.

row is an `INTENT(INOUT)` rank-1 array of type `INTEGER` and size at least `ptr(n+1)-1`. It must hold the row indices of the entries of the lower triangular part of K with the row indices for the entries in column 1 preceding those for column 2, and so on. Within each column, the row indices must be in increasing order (so that all entries of the column in the $(1,1)$ block A must precede those in the $(2,2)$ block $-C$). The diagonal entry in the first $n-m$ columns must be present (but may have value zero). If `control%check` is set to `.true.` (the default), `row` is checked for errors and duplicates and out-of-range indices are removed; otherwise, `row` is unchanged.

val is an `INTENT(INOUT)` rank-1 array of package type and size at least `ptr(n+1)-1`. `val(k)` must hold the value of the entry in `row(k)`. If `control%check` is set to `.true.` (the default), on exit duplicates are summed and out-of-range indices removed; otherwise, `val` is unchanged.

lsize is an `INTENT(IN)` scalar of type `INTEGER` that determines the maximum number of fill entries within each column of the incomplete factor L. In general, increasing `lsize` improves the quality of the preconditioner but increases the time to compute and then apply the preconditioner (see Section 4). Values less than `0` are treated as `0`.

rsize is an `INTENT(IN)` scalar of type `INTEGER` that determines the maximum number of entries within each column of the strictly lower triangular matrix R that is used in the computation of the preconditioner. A rank-1 array of type `INTEGER` and a rank-1 array of package type each of size rsize$\times$n are allocated internally to hold R. Thus the amount of memory used, as well as the amount of work involved in computing the preconditioner, depends on `rsize`. Setting `rsize` > 0 generally leads to a higher quality preconditioner than using `rsize` $= 0$ (and `rsize` $\geq$ `lsize` is generally recommended). Values less than `0` are treated as `0`.

keep is an INTENT(OUT) scalar of type mi30_keep. It is used to hold data about the problem being solved and must be passed unchanged to mi30_precondition. The following components may be of interest to the user:

>   fact_dinv is an allocatable rank-1 array of package type. On exit, it is allocated to have size n and fact_dinv(k) holds the value of the k-th diagonal entry of D^{-1}.
>
>   fact_ptr is an allocatable rank-1 array of type INTEGER(long). On exit, it is allocated to have size n+1, fact_ptr(j) holds the position in fact_row of the first entry in column j of the computed factor L and ptr(n+1) is set to one more than the number of entries in L.
>
>   fact_row is an allocatable rank-1 array of type INTEGER. On exit, the first fact_ptr(n+1)-1 entries hold the row indices of the entries of the computed factor L, with the row indices for the entries in column 1 preceding those for column 2, and so on.
>
>   fact_val is an allocatable rank-1 array of package type. On exit, fact_val holds the values of the entries in the computed factor L such that fact_val(k) is the value of the entry in fact_row(k).
>
>   scale is an allocatable rank-1 array of package type. On exit, if control%iscale $>$ 0, it is allocated to have size n and holds the scaling factors for A.
>
>   invp is an allocatable rank-1 array of type INTEGER. On exit, if control%iorder $>$ 0, it is allocated to have size n and specifies the permutation such that the j-th column of the permuted matrix $Q^T K Q$ is the invp(j)-th column of K (that is, invp(j) is the index of the j-th pivot).
>
>   perm is an allocatable rank-1 array of type INTEGER. On exit, if control%iorder $>$ 0, it is allocated to have size n and specifies the elimination ordering such that perm(i) holds the position of i-th column of K in the elimination order.

control is an INTENT(IN) scalar of type mi30_control (see Section 2.5).

info is an INTENT(OUT) scalar of type mi30_info. Its components provide information about the execution of the subroutine, as explained in Section 2.6.

scale is an optional INTENT(IN) rank-one array of package type and size n that must be present if control%order=5. In this case, scale must be set by the user to hold scaling factors for A.

perm is an optional INTENT(IN) rank-one array of type INTEGER and size n that must be present if control%iorder=3. In this case, the user must supply an elimination ordering such that perm(i) holds the position of the i-th column of K in the elimination order. A column with index $j > n - m$ should only be ordered after all the columns with index $i \leq n - m$ for which $k_{ij} \neq 0$ have been ordered; if this condition is not satisfied, a modified ordering will be used (and returned in keep%perm).

2.4.4   To perform preconditioning operations

The signed incomplete Cholesky factorization preconditioner may be applied to compute $y = Pz$ by making a call as follows.

```
    call mi30_precondition(job, n, keep, z, y, info)
```

job is an INTENT(IN) scalar of type INTEGER that must be set as follows:

>   1 if $P = (\overline{L} D \overline{L}^T)^{-1}$ is to be used as the preconditioner ($\overline{L} = QS^{-1}L$).
>
>   2 if $P = (\overline{L} |D| \overline{L}^T)^{-1}$ is to be used as the preconditioner.

n, keep: must be unchanged since the call to mi30_factorize.

z is an INTENT(IN) rank-1 array of package type and size n. It must be set by the user to hold the vector z to which the incomplete factorization preconditioner P is to be applied.

y is an INTENT(OUT) rank-1 array of package type and size n. On exit, y contains Pz.

info is an INTENT(INOUT) scalar of type mi30_info. Only the component info%flag is accessed (see Section 2.6).

2.4.5 To perform solve operations

The system $\overline{L}\,Dy = SQ^T z$, or $\overline{L}\,|D|y = SQ^T z$ or $\overline{L}^T\,S^{-1}Q^T y = z$ may be solved by making a call as follows.

```
call mi30_solve(job, n, keep, z, y, info)
```

job is an INTENT(IN) scalar of type INTEGER that must be set as follows:

> 1 if the solution of $\overline{L}\,Dy = SQ^T z$ is required,
>
> 2 if the solution of $\overline{L}\,|D|y = SQ^T z$ is required,
>
> 3 if the solution of $\overline{L}^T\,S^{-1}Q^T y = z$ is required.

n, keep: must be unchanged since the call to mi30_factorize.

z is an INTENT(IN) rank-1 array of package type and size n. It must be set by the user to the right-hand side vector z.

y is an INTENT(OUT) rank-1 array of package type and size n. On exit, y contains the solution vector y.

info is an INTENT(INOUT) scalar of type mi30_info. Only the component info%flag is accessed (see Section 2.6).

2.4.6 The finalisation subroutine

Once all other calls are complete for a problem or after an error return, a call should be made to free memory allocated by hsl_mi30_factorize using a call to mi30_finalise.

```
call mi30_finalise(keep, info)
```

keep is an INTENT(INOUT) scalar of type mi30_keep that must be passed unchanged. On exit, allocatable components will have been deallocated.

info is an INTENT(INOUT) scalar of type mi30_info. Only the components info%flag and info%stat are accessed (see Section 2.6).

2.5 The control derived data type

The derived data type mi30_control is used to hold controlling data; it is used by mi30_factorize only. The components are automatically given default values in the definition of the type.

Components that control printing

unit_error is a scalar of type INTEGER with default value 6 that is used as the output stream for error messages. If it is negative, these messages will be suppressed.

unit_warning is a scalar of type INTEGER with default value 6 that is used as the output stream for warning messages. If it is negative, these messages will be suppressed.

Components that control the initial and subsequent choice of the shifts $\alpha(1:2)$.

Note that the aim is to choose the shifts to be as small as possible to avoid breakdown of the Cholesky factorization process (see Section 4).

alpha is a rank-1 array of package type and size 2 with default values $(0.0, 0.0)$ that holds the initial shifts $\alpha(1:2)$. Values less than zero are treated as zero.

lowalpha is a scalar of package type with default value 0.001 that controls the choice of the shift in the event of a breakdown. Values less than or equal to zero are treated as the default.

maxshift is a scalar of type INTEGER with default value 3 that controls the maximum number of times the shift can be decreased after a successful factorization with a positive shift. See Section 4 for details. Limiting maxshift may reduce the factorization time but may result in a poorer quality preconditioner.

shift_factor is a scalar of package type with default value 4.0 that controls how rapidly a shift is increased after a breakdown. See Section 4 for details. Increasing shift_factor may reduce the factorization time but may result in a poorer quality preconditioner. Values less than one are treated as the default.

shift_factor2 is a scalar of package type with default value 4.0 that controls how rapidly a shift is decreased after a successful factorization with a positive shift. See Section 4 for details. Values less than one are treated as the default.

small is a scalar of type REAL. Any pivot whose modulus is less than small is treated as zero and, if such a pivot is encountered, the factorization breaks down, a shift is increased and the factorization restarted. The default in the double version is 10^{-20} and in the single version is 10^{-12}.

Components that control the dropping of small entries

tau1 and tau2 are scalars of package type with default values 0.001 and 0.0001. They control the dropping of entries from L and R. In the computation of the incomplete factorization, entries of magnitude less than |tau1| are dropped from L; those that are at least |tau2| but less than |tau1| may be included in R while those less than |tau2| are dropped from R.

Other components

check is a scalar of type LOGICAL with default value .true.. If .true., the matrix data is checked for errors and the cleaned matrix (duplicates are summed, out-of-range entries discarded and, within each column, the entries are ordered by increasing row index) overwrites the user-supplied data in ptr, row and val. Otherwise, no checking of the matrix data is carried out (it is important to note that any out-of-order entries or out-of-range entries or duplicates may cause HSL_MI30 to fail in an unpredictable way) and so it is recommended that the matrix data is checked.

iorder is a scalar of type INTEGER with default value 6 that indicates the ordering that is required. The chosen ordering is computed and then post-processed (see Section 4). Options available are:

$\leq$ 0 no ordering.

1 A reverse Cuthill-McKee (RCM) ordering (computed using MC61) is used.

2 An approximate minimum degree (AMD) ordering (computed using HSL_MC68) is used.

3 User-supplied ordering is used.

4 The rows are ordered by ascending degree.

5 METIS (nested dissection) ordering with default settings is used. If METIS is not supplied and this option is requested, the routine will return immediately with an error.

6 A Sloan profile reduction ordering (computed using MC61) is used. This is the default.

If iorder>6, the default is used.

iscale is a scalar of type INTEGER with default value 1 that indicates the scaling that is required. Options available are:

≤ 0 No scaling.

1 Scaling generated using the l_2-norm of the columns of A. This is the default.

2 Scaling generated by applying the iterative method of the package MC77 for one iteration in the infinity norm and three iterations in the one norm (equilibration ordering).

3 Scaling generated from a weighted bipartite matching using the package HSL_MC64.

4 Diagonal scaling is used.

5 User-supplied scaling is used. The user must supply scaling factors for A.

If iscale>5, the default is used.

rrt is a scalar of type LOGICAL with default value .false. that is used to control whether the entries of RR^T (see (1.2)) that cause no additional fill-in in (1.1) are allowed ($rsize > 0$ only). Allowing such entries can improve the quality of the preconditioner (although this is not guaranteed) but at some additional computational cost in the factorization process. If rrt=.true. such entries are allowed; otherwise, they are not allowed.

2.6 The derived data type for holding information

The derived data type mi30_info is used to hold information from the execution of mi30_factorize. The components are:

alpha is a rank-1 array of package type and size 2 that holds the final shifts (it is set to zero if no shifts are used).

band_after is a scalar of type INTEGER. If control%iorder=1 or 6, band_after holds the semibandwidth of A after reordering; otherwise, it is set to 0.

band_before is a scalar of type INTEGER. If control%iorder=1 or 6, band_before holds the semibandwidth of A before reordering; otherwise, it is set to 0.

dup is a scalar of type INTEGER that holds the number of duplicated indices removed from row.

flag is a scalar of type INTEGER that gives the exit status of the algorithm (details in Section 2.7).

flag61 is a scalar of type INTEGER that holds the exit status on return from MC61 (and is set to 0 if MC61 is not used).

flag64 is a scalar of type INTEGER that holds the exit status on return from HSL_MC64 (and is set to 0 if HSL_MC64 is not used).

flag68 is a scalar of type INTEGER that holds the exit status on return from HSL_MC68 (and is set to 0 if HSL_MC68 is not used).

flag77 is a scalar of type INTEGER that holds the exit status on return from MC77 (and is set to 0 if MC77 is not used).

nrestart is a scalar of type INTEGER that holds the number of restarts (after reducing a shift).

nshift is a scalar of type INTEGER that holds the number of non-zero shifts used.

oor is a scalar of type INTEGER that holds the number of out-of-range indices removed from row.

profile_after is a scalar of package type. If control%iorder=1 or 6, profile_after holds the profile of *A* after reordering; otherwise, it is set to 0.0.

profile_before is a scalar of package type. If control%iorder=1 or 6, profile_before holds the profile of *A* before reordering; otherwise, it is set to 0.0.

size_r is a scalar of type INTEGER(long) that holds the size of the integer and real arrays that are used during the factorization to hold *R*.

stat is a scalar of type INTEGER that holds the Fortran stat parameter.

2.7 Warning and error messages

A successful return from a subroutine in the package is indicated by info%flag having the value zero. A negative value is associated with an error message that by default will be output on unit control%unit_error.

Possible negative values are:

−1 memory allocation failed. The stat parameter is returned in info%stat.

−2 The array row is too small.

−3 The array val is too small.

−4 Either n or m is out of range (n<2, m<1 or m>n/2).

−5 Error in the array ptr.

−6 One or more diagonal entries in the $(1, 1)$ *A*-block is missing.

−7 Unexpected error returned by MC77. The MC77 exit status is returned in info%flag77.

−8 Unexpected error returned by HSL_MC64. The HSL_MC64 exit status is returned in info%flag64.

−9 HSL_MC64 has found that *K* is structurally singular.

−10 The optional argument scale is not present when it should be.

−11 The optional argument perm is either not present when it should be or it does not hold a permutation.

−12 Unexpected error returned by MC61. The MC61 exit status is returned in info%flag61. Note that, if the matrix has not been checked for errors and there are duplicated or out-of-range entries in row, mc61 will return an error flag of −4 and the computation will terminate.

−13 Unexpected error returned by HSL_MC68. The HSL_MC68 exit status is returned in info%flag68. Note that this error is returned if METIS ordering has been requested (control%iorder=5) but METIS is not linked).

−14 Memory deallocation failed. The stat parameter is returned in info%stat.

−15 All entries in one or more columns are out of range.

−16 One or more of the diagonal entries of the $(2, 2)$ block $−C$ is positive.

Positive values for `info%flag` are associated with a warning and can only be returned by `mi30_factorize`. Possible positive values are:

+1 Out-of-range indices have been removed from `row`. The number of such entries is given in `info%oor`.

+2 Duplicated entries were found in `row`; these have been removed and the corresponding entries in `val` have been summed. The number of such entries is given in `info%dup`.

+3 A warning has been issued by `HSL_MC64` that the computed scaling factors are large and may cause overflow when used to scale the matrix. No scaling is used.

+4 A warning has been issued by `MC61`. The MC61 exit status is returned in `info%flag61`.

3 GENERAL INFORMATION

Input/output: Error messages on unit `control%lp` and warning and diagnostic messages on units `control%wp` and `control%mp`, respectively. These have default value 6; printing of these messages is suppressed if the relevant unit number is negative or if `print_level` is negative.

Restrictions: 1≤m<n.

4 METHOD

`mi30_factorize` starts by optionally checking the matrix data for errors; this is done using `HSL_MC69`. Checking removes out-of-range entries, sums duplicates, and reorders the entries within each column by increasing row index. A scaling and/or ordering is then optionally computed; HSL packages are used to do this. Unless a problem is known to be well scaled, **scaling is highly recommended**. We impose a constraint on the ordering: a pivot corresponding to a variable i in the $(2,2)$ block $-C$ can only be eliminated once all the variables that corresponding to the entries in column i with row index $j \leq n-m$ have been eliminated (in graph terms, a C-node can only be eliminated once all its A-node neighbours have been eliminated). Thus once an ordering has been computed using, for example, the Sloan algorithm, it is modified to satisfy the above constraint before the factorization begins.

A left-looking sparse Cholesky algorithm is used to compute the signed incomplete factorization, one column at a time. The parameters `lsize` and `rsize` control the amount of memory used as well as the amount of work involved in computing the factorization. `lsize` controls the number of entries in the computed incomplete factor L (at most `lsize` fill entries are permitted in each column of L) and `rsize` limits the number of entries in each column of the matrix R. If `rsize=0` and `control%tau1=0.0`, the incomplete factorization is essentially that of [1]. However, it is generally advantageous (in terms of the quality of the preconditioner) to use `rsize>0`. Increasing `lsize` and/or `rsize` increases the cost of the factorization (in terms of time and memory). Furthermore, increasing `lsize` leads to a denser incomplete factorization (but one that is, in general, a better preconditioner), increasing the cost of each call to `mi30_precondition` and `mi30_solve`. Values of `lsize` and `rsize` equal to 10 is often a reasonable choice but, if the preconditioner is to be used for many problems, it may be worthwhile to experiment with a range of values to try and get the best overall performance; smaller values may be used if the memory available is limited or larger values may be used to try and obtain a higher quality preconditioner.

Dropping parameters `control%tau1` and `control%tau2` are used to further sparsify L and R, respectively. As each column of L is computed, entries of absolute value less than `control%tau1` are dropped. These may be included in R but entries less than `control%tau2` are dropped from R.

In the event of breakdown within the factorization (that is, a pivot is encountered that is smaller in absolute value than `control%small`), a diagonal shift is used. If the breakdown occurs for a pivot in the A-block, a shift $\alpha(1)$ is used; if breakdown occurs in the $(2,2)$ block $-C$, a shift $-\alpha(2)$ is used ($\alpha(1:2) > 0$). It is important to try and use

as small a shift as possible but also to limit the number of breakdowns. The user can supply initial shifts $\alpha_0(1:2)$. If breakdown occurs, a larger shift

$$\alpha_1(j) = \max(\texttt{control\%lowalpha}, \ \alpha_0(j) \times \texttt{control\%shift\_factor}),$$

is tried, where $j = 1$ if breakdown is in the A-block and $j = 2$ otherwise. The process continues until an incomplete factorization of

$$\bar{K} = SQ^T \left(\begin{array}{cc} A & B^T \\ B & -C \end{array} \right) QS + \left(\begin{array}{cc} \alpha(1)I & 0 \\ & -\alpha(2)I \end{array} \right)$$

is successful. If breakdown occurs at the same (or nearly the same) stage of the factorization for two successive shifts, to try and limit the number of restarts, $\alpha(j)$ is increased by a factor of $2 \times \texttt{control\%shift\_factor}$. Conversely, if $\alpha_k(j) = \texttt{control\%lowalpha(j)}$, to prevent an unnecessarily large shift from being used, the shift is decreased by setting

$$\alpha_{k+1}(j) = \alpha_k(j)/\texttt{control\%shift\_factor2},$$

and applying the incomplete factorization algorithm to

$$\bar{K}_{k+1} = SQ^T \left(\begin{array}{cc} A & B^T \\ B & -C \end{array} \right) QS + \left(\begin{array}{cc} \alpha_{k+1}(1)I & 0 \\ & -\alpha_{k+1}(2)I \end{array} \right).$$

If this factorization is also breakdown free, the process is repeated (up to $\texttt{control\%maxshift}$ times). In all cases, the values of the final shifts are returned to the user in $\texttt{info\%alpha(1:2)}$, along with the number of shifts tried and the number of restarts ($\texttt{info\%nrestart}$).

For further details, see [1] and [2].

References:

[1] J. A. Scott and M. Tůma. (2013). `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. RAL Technical Report. RAL-P-2013-004. See also ACM Trans. Math. Software 40 (2014), 24:1–24:19.

[2] J. A. Scott and M. Tůma. (2014). On signed incomplete Cholesky factorization preconditioners for saddle-point systems. RAL Technical Report. RAL-P-2014-003. See also SIAM J. Sci. Computing 36 (2014), A2984–A3010.

5 EXAMPLE OF USE

Suppose we wish to use preconditioned GMRES to solve the linear system $Kx = b$ with

$$K = \left(\begin{array}{cccc} 4 & 0 & 1 & -1 \\ 0 & 3 & 0 & 2 \\ 1 & 0 & 4 & 0 \\ -1 & 2 & 0 & -1 \end{array} \right) \quad \text{and} \quad b = \left(\begin{array}{c} 4 \\ 5 \\ 5 \\ 0 \end{array} \right).$$

We may use the following code:

```
program mi30_spec_double

   use hsl_mi30_double
   implicit none
```

```
integer, parameter :: wp = kind(1.0d0)
integer, parameter :: long = selected_int_kind(18)

type(mi30_control) :: control
type(mi30_info) :: info
type(mi30_keep) :: keep

integer,  allocatable :: ptr(:),row(:)
real(wp), allocatable :: h(:,:), val(:), w(:,:)

! Arrays and scalars required by the GMRES code mi24
real(wp) :: resid
real(wp) :: cntl24(4),rsave24(9)
integer  :: icntl24(8),isave24(17),info24(4)
logical  :: lsave24(4)

integer :: iact,locy,locz,lsize,m,m_restart,n,nz,rsize

! Read in the matrix data
read (5,*) n,m,nz

! Choose restart parameter for GMRES
m_restart = 10

! Allocate arrays for matrix and for GMRES
allocate (ptr(n+1),row(nz),val(nz), &
          w(n,m_restart+7),h(m_restart+1,m_restart+2))
read (5,*) ptr(1:n+1)
read (5,*) row(1:nz)
read (5,*) val(1:nz)
read (5,*) w(1:n,1) ! Right-hand side array

! Choose lsize and rsize
lsize = 1
rsize = 1

control%iorder = 0 ! use supplied order
control%iscale = 0 ! do not scale

! Compute the preconditioner
call mi30_factorize(n, m, ptr, row, val, lsize, rsize, keep, control, info)

if (info%flag.lt.0) then
  write (*,'(a,i4)') ' Unexpected error from mi30_factorize. flag = ',info%flag
  call mi30_finalise(keep,info)
  stop
end if

! Prepare to use the GMRES code mi24 with preconditioning
```

```
    call mi24id(icntl24, cntl24, isave24, rsave24, lsave24)
    icntl24(3) = 2 ! right preconditioning

    iact = 0
    do
        call MI24AD(iact, n, m_restart, w, size(w, 1), locy, locz, h, &
            size(h, 1), resid, icntl24, cntl24, info24, isave24,   &
            rsave24, lsave24)

      select case(iact)
      case(-11) ! Error
         write (*,'(a,i4)') ' Unexpected error from mi24. flag = ',info24(1)
         exit

      case(1) ! convergence achieved
         write (*,'(a,i3,a)') &
           ' GMRES Convergence in ',info24(2),' iteration(s)'
         write (*,'(a)') ' Solution = '
         write (*,'(5es12.4)') w(1:n,2)
         exit

       case(2) ! Form y = Kz
          call mxmult(n,ptr,row,val,w(1:n,locz),w(1:n,locy))

       case(4) ! Preconditioner
          call mi30_precondition(1, n, keep, w(1:n,locz), w(1:n,locy), info)
      end select

    end do
    call mi30_finalise(keep,info)

contains
!*************************************************************
!    sparse matrix-vector multiplication y=K*x.
!    Lower triangle of sparse matrix K held.

    subroutine mxmult(n,ptr,row,val,x,y)

    real(wp), parameter :: zero = 0.0_wp

    integer, intent(in) :: n
    integer, intent(in) :: ptr(n+1),row(:)
    real(wp), intent(in) :: val(:),x(n)
    real(wp), intent(out) :: y(n)

    integer:: i,j,k
    real(wp) :: sum

    y = zero
```

```
      do i = 1,n
        sum = zero
        do j = ptr(i),ptr(i+1)-1
           k = row(j)
           if (k.ne.i) y(k) = y(k) + val(j)*x(i)
           sum = sum + val(j)*x(k)
        end do
        y(i) = y(i) + sum
      end do

      end subroutine mxmult

end program mi30_spec_double
```

With the input data:

```
 4  1  7
 1  4  6  7  8
 1  3  4  2  4  3  4
 4.  1.  -1.  3.  2.  4.  -1.
 4.  5.  5.  0.
```

we obtain the following output:

```
 GMRES Convergence in   1 iteration(s)
 Solution =
  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
```