Science and Technology Facilities Council

HSL_MC81

PACKAGE SPECIFICATION HSL

1 SUMMARY

HSL_MC81 uses randomized algorithms to compute low-rank approximations of a given matrix. If A is $n \times n$ and symmetric positive definite it computes a truncated eigenvalue decomposition (EVD)

$$A \approx XDX^T,$$

where $k < n$ is the target rank, X is an $n \times k$ orthonormal matrix and D is a $k \times k$ diagonal matrix whose entries approximate the eigenvalues of A with largest absolute value in descending order.

If A is a general $m \times n$ matrix, it computes a truncated singular value decomposition (SVD)

$$A \approx U\Sigma V^T,$$

where $k < \min(m,n)$ is the target rank, U is an $m \times k$ orthonormal matrix, V is an $n \times k$ orthonormal matrix and Σ is a $k \times k$ diagonal matrix whose entries approximate the largest singular values of A in descending order.

The package offers simple interfaces in which the user supplies the sparse matrix A in compressed sparse column (CSC) format and, for greater flexibility, matrix-free reverse communication interfaces that return control to the user when products with A are required.

ATTRIBUTES — Version: 1.0.0 (11 January 2022). **Interfaces:** Fortran, MATLAB. **Types:** Real (single, double). **Original date:** January 2022. **Uses:** HSL_FA14, KB08, BLAS subroutines _gemm, _gemv, _trsm, and the LAPACK subroutines _gesvd, _gemqrt, _geqrt, _potrf, _syev. **Origin:** J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement of the form

Single precision version

```
USE HSL_MC81_single
```

Double precision version

```
USE HSL_MC81_double
```

If it is required to use more than one module at the same time, the derived types (Section 2.2) must be renamed in one of the use statements.

The following subroutines may be called by the user.

Standard interface

(a) MC81_revd: given the lower triangular part of the sparse matrix A in compressed sparse column format, this routine computes a truncated EVD.

(b) MC81_rsvd: given the sparse matrix A in compressed sparse column format, this routine computes a truncated SVD.

Reverse communication interface

(a) MC81_revd_reverse: matrix-free computation of a truncated EVD.

(b) MC81_rsvd_reverse: matrix-free computation of a truncated SVD.

In addition, MC81_print_message can be used after a return from any of the above routines to print the error message associated with a non zero error flag.

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 MC81_info, MC81_control and MC81_keep (reverse communication interface only). The following pseudocode illustrates this.

```
use HSL_MC81_double
...
type (MC81_control) :: control
type (MC81_info) :: info
type (MC81_keep) :: keep
```

The components of MC81_control and MC81_info are explained in Sections 2.4.1 and 2.4.2. The components of MC81_keep are private and are used for communication between calls when using the reverse communication interface.

2.3 Argument lists and calling sequences

2.3.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that optional arguments be called by keyword, not by position**.

2.3.2 Integer, real and package types

INTEGER denotes default integer. 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.

2.3.3 Input of the matrix A

If the standard interface is used, the user must supply the matrix A in compressed sparse column format (with only the **lower** triangular part of A supplied for the EVD routine). **No checks** are made on the user's data (apart from scalar arguments). It is important to note that any out-of-range entries or duplicates may cause HSL_MC81 to fail in an unpredictable way. Before using `HSL_MC81`, the HSL package `HSL_MC69` may be used to check for errors and to handle duplicates (`HSL_MC69` sums them) and out-of-range entries (`HSL_MC69` removes them).

If the user's data is held using another standard sparse matrix format (such as coordinate format or sparse compressed row format), we recommend using a conversion routine from `HSL_MC69` to put the data into compressed sparse column format.

If the reverse communication interface is used, the matrix A need not be held explicitly. The user only needs to be able to form products with A and A^T.

2.3.4 Randomized EVD: standard interface

For a **sparse symmetric positive definite** $n \times n$ matrix A with the lower triangular part held in compressed sparse column format a rank-k EVD may be computed using a call of the form:

```
call MC81_revd(n, ptr, row, val, k, p, d, x, control, info [,algorithm])
```

`n` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the order of A. **Restriction:** `n` ≥ 0.

`ptr` is an `INTENT(IN)` rank-one array of type `INTEGER` and size `n+1` that must be set by the user so that `ptr(j)` is the position in array `row` of the first entry in column `j` (`j=1,2,...,n`) and `ptr(n+1)` must be set to one more than the total number of entries in the lower triangular part of A.

`row` is an `INTENT(IN)` rank-one array of type `INTEGER`. It must be set so that `row(1:ptr(n+1)-1)` holds the row indices of the non zero entries in the **lower triangular** part of A. The entries of a single column must be contiguous, with the diagonal entry held as the **first entry** in the column and there must be no null columns. The entries of column `j` must precede those of column `j+1` (`j=1,2,...,n-1`), and there must be no wasted space between columns. There must be no out-of-range entries or duplicated entries.

`val` is an `INTENT(IN)` rank-one array of package type. It must be set so that `val(k)` holds the value of the entry in `row(k)` (`k=1,2,...,ptr(n+1)-1`).

`k` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the target rank. **Restriction:** `k` > 0.

`p` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the oversampling parameter (see Section 4). Increasing `p` increases the computational costs but can lead to more accurate approximate eigenpairs. **Restrictions:** `p`$\geq$`0`; $\mathtt{k}+\mathtt{p}<\mathtt{n}$.

`d` is an `INTENT(OUT)` rank-one array of type **package type** of size `k+p`. On successful exit, `d(1:k)` holds approximate eigenvalues of A in decreasing order of their absolute values.

`x` is an `INTENT(INOUT)` rank-two array of package type with extents at least `n` and `2(k+p)`. If `control%in_G = .true.`, `x(1:n,1:k+p)` must be set by the user to hold a random matrix. On successful exit, `x(1:n,1:k)` holds approximate eigenvectors of A corresponding to the approximate eigenvalues in `d(1:k)`.

`control` is a scalar `INTENT(IN)` argument of type `mc81_control`. Its components control the execution of the subroutine, as explained in Section 2.4.1.

`info` is a scalar `INTENT(OUT)` argument of type `mc81_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.4.2.

`algorithm` is an optional scalar of type `INTEGER` that controls which algorithm is used and, in particular, whether Nyström's method is used (`algorithm = 2`). Nyström's method requires A to be positive definite and, in this case, it generally computes higher quality eigenpairs at almost no additional cost and so is recommended. For all other values of `algorithm` (and if `algorithm` is not present), Nyström's method is not used. Further details are given in Section 4.

2.3.5 Randomized EVD: reverse communication interface

For an $n \times n$ **symmetric positive definite** matrix A, a rank-k EVD may be computed using calls of the form:

```
call MC81_revd_reverse(action, n, k, p, d, x, keep, control, info [,algorithm])
```

`action` is a scalar `INTENT(INOUT)` argument of type `INTEGER` that controls the task that must be performed by the user. It must be set to `0` before the first call and must then not be changed by the user. Possible values are:

- `0` The computation has terminated. If `info%flag= 0`, the computation was successful. Check `info%flag` for other reasons for terminating (see Section 2.5).
- `1` The user must compute `x(1:n,k+p+1:2(k+p))` $=A$ `x(1:n,1:k+p)` (without altering `x(1:n,1:k+p)`) and then recall `mc81_revd_reverse`. All other arguments must be unchanged.

`n, k, p, control, info, algorithm` are as in Section 2.3.4, with the exception that `info` is `INTENT(INOUT)`. These arguments must be unchanged by the user between calls.

`d` is an `INTENT(INOUT)` rank-one array of type **package type** of size `k+p`. It need not be set prior to the first call. On successful exit with `action=0`, `d(1:k)` holds approximate eigenvalues of A in decreasing order of their absolute values.

`x` is an `INTENT(INOUT)` rank-two array of package type with extents at least `n` and `2(k+p)`. If `control%in_G=`.true., prior to the first call, `x(1:n,1:k+p)` must be set by the user to hold a random matrix. On each exit with `action=1`, the columns of `x` determined by `pos` must be multiplied by A. On successful exit with `action=0`, `x(1:n,1:k)` holds approximate eigenvectors of A corresponding to the approximate eigenvalues in `d(1:k)`.

`keep` is a scalar `INTENT(INOUT)` argument of type `mc81_keep`. It must be unchanged by the user between calls.

2.3.6 Randomized SVD: standard interface

For a general sparse matrix $A \in \mathbb{R}^{m\times n}$ in compressed sparse column format a rank-k SVD may be computed using a call of the form:

```
call MC81_rsvd(m, n, ptr, row, val, k, p, s, u, vt, g, q, control, info)
```

`m` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the number of rows in A. **Restriction:** `m` ≥ 0.

`n` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the number of columns in A. **Restriction:** `n` ≥ 0.

`ptr` is an `INTENT(IN)` rank-one array of type `INTEGER` and size `n+1` that must be set by the user so that `ptr(j)` is the position in the array `row` of the first entry in column `j` (`j=1,2,...,n`) and `ptr(n+1)` must be set to one more than the total number of entries.

`row` is an `INTENT(IN)` rank-one array of type `INTEGER`. It must be set so that `row(1:ptr(n+1)-1)` holds the row indices of the entries of A. The entries of a single column must be contiguous. The entries of column `j` must precede those of column `j+1` (`j=1,2,...,n-1`) and there must be no wasted space between columns. Row indices within a column may be in any order. There must be no out-of-range entries or duplicated entries.

`val` is an `INTENT(IN)` rank-one array of package type. It must be set so that `val(k)` holds the value of the entry in `row(k)` (`k=1,2,...,ptr(n+1)-1`).

`k` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the target rank. **Restriction:** `k` $>$ `0`.

`p` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the oversampling parameter (see Section 4). Increasing `p` increases the computational costs but can lead to more accurate singular values and vectors. **Restriction:** `p`$\geq$`0`; $\mathtt{k}+\mathtt{p}<\mathtt{n}$.

`s` is an `INTENT(OUT)` rank-one array of type **package type** of size `k+p`. On successful exit, `s(1:k)` holds approximate singular values of A in decreasing order.

`u` is an `INTENT(OUT)` rank-two array of package type with extents at least `m` and `k+p`. On successful exit, `u(1:m,1:k)` holds approximate left singular vectors of A corresponding to the approximate singular values in `s(1:k)`.

`vt` is an `INTENT(OUT)` rank-two array of package type with extents at least `k+p` and `n`. On successful exit, `vt(1:k, 1:n)` holds approximate right singular vectors of A **stored row wise** corresponding to the approximate singular values in `s(1:k)`.

`g` is an `INTENT(INOUT)` rank-two array of package type with extents at least `n` and `k+p`. If `control%iters`$\geq$`0` and `control%in_G=.true.`, then `g(1:n,1:k+p)` must be set by the user to hold a random matrix.

`q` is an `INTENT(INOUT)` rank-two array of package type with extents at least `m` and `k+p`. If `control%iters<0` and `control%in_G=.true.`, then `q(1:m,1:k+p)` must be set by the user to hold a random matrix.

`control` is a scalar `INTENT(OUT)` argument of type `mc81_control`. Its components control the execution of the subroutine, as explained in Section 2.4.1.

`info` is a scalar `INTENT(OUT)` argument of type `mc81_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.4.2.

2.3.7 Randomized SVD: reverse communication interface

For a general matrix $A\in\mathbb{R}^{n\times n}$, a rank-$k$ SVD may be computed using calls of the form:

```
call MC81_rsvd_reverse(action, m, n, k, p, s, u, vt, g, q, b, w, keep, control, info)
```

`action` is a scalar `INTENT(INOUT)` argument of type `INTEGER` that controls the task that must be performed by the user. It must be set to `0` before the first call. Possible values are:

- `0` The computation has terminated. If `info%flag= 0` the computation was successful. Check `info%flag` for other reasons for terminating (see Section 2.5).
- `1` The user must compute `u(1:m,1:k+p)`$=AG$, with G held in `g(1:n,1:k+p)` and then recall `mc81_rsvd_reverse`. All other arguments must be unchanged.
- `2` The user must compute `b(1:k+p,1:n)`$=Q^TA$, with Q held in `q(1:m,1:k+p)` and then recall `mc81_rsvd_reverse`. All other arguments must be unchanged.
- `3` The user must compute `g(1:n,1:k+p)`$=A^TQ$, with Q held in `q(1:m,1:k+p)` and then recall `mc81_rsvd_reverse`. All other arguments must be unchanged.
- `4` The user must compute `u(1:m,1:k+p)`$=AW$, with W held in `w(1:n,1:k+p)` and then recall `mc81_rsvd_reverse`. All other arguments must be unchanged.

`m, n, k, p, control, info` are as in Section 2.3.6, with the exception that `info` is `INTENT(INOUT)`. These arguments must be unchanged by the user between calls.

`s` is an `INTENT(INOUT)` rank-one array of type **package type** of size `k+p`. It need not be set prior to the first call. On successful exit with `action=0`, `s(1:k)` holds approximate singular values of A in decreasing order.

u is an INTENT(INOUT) rank-two array of package type with extents at least m and k+p. It need not be set prior to the first call. If control%iters<0 and control%in_G=.true., prior to the first call, q(1:m,1:k+p) must be set by the user to hold a random matrix. Otherwise, it need not be set initially but must be used by the user on returns with action = 4. On successful exit with action=0, the first k columns hold approximate left singular vectors of *A* corresponding to the approximate singular values in s(1:k).

vt is an INTENT(INOUT) rank-two array of package type with extents at least k+p and n. It need not be set prior to the first call. On successful exit with action=0, the first *k* rows hold approximate right singular vectors of *A* **stored row wise** corresponding to the approximate singular values in s(1:k).

g is an INTENT(INOUT) rank-two array of package type. If control%iters≥0 then it must have extents at least n and k+p; otherwise it is not accessed. If control%iters≥0 and control%in_G=.true., prior to the first call, g(1:n,1:k+p) must be set by the user to hold a random matrix. It must be used by the user on returns with action = 1 and 3.

q is an INTENT(INOUT) rank-two array of package type with extents at least m and k+p. It should not be set prior to the first call. It must be used by the user on returns with action = 1, 2 and 3.

b is an INTENT(INOUT) rank-two array of package type with extents at least k+p and n. It should not be set prior to the first call. It must be used by the user on returns with action = 2.

w is an INTENT(INOUT) rank-two array of package type with extents at least n and k+p. It should not be set prior to the first call. It must be used by the user on returns with action = 4.

keep is a scalar INTENT(OUT) argument of type mc81_keep. It must be unchanged by the user between calls.

2.4 Printing error messages

After a return from any of the other routines, a call of the following form may be made to print an error message associated with a non zero error flag info%flag.

```
call MC81_print_message(info,[unit, message])
```

info is a scalar INTENT(IN) argument of type mc81_info. It must be unchanged since the call that returned an error.

unit is an optional scalar INTENT(IN) argument of type INTEGER that holds the unit number for printing the error message. If it is negative, printing is suppressed. If unit is not present, the message will be printed on unit 6.

message is an optional scalar INTENT(IN) argument of type character(len=*). If present, on entry, it must hold the message to be printed ahead of the error message.

2.4.1 The derived data type for controlling the execution

The derived data type mc81_control is used to control the execution. The components are:

in_G is scalar of type LOGICAL that controls whether the user supplies a random matrix *G*. If in_G is set to .true. and an EVD is to be computed, then x(1:n,1:k+p) must set set by the user to hold *G* prior to calling mc81_revd (or prior to the first call to mc81_revd_reverse). If it is set to .true. and an SVD is to be computed, then if control%iters≥0 (respectively, control%iters<0) g(1:n,1:k+p) (respectively, q(1:m,1:k+p)) must set set by the user to hold *G* prior to calling mc81_rsvd (or prior to the first call to mc81_rsvd_reverse). Otherwise, *G* is generated by the package. The default value is .false..

`iters` is a scalar of type `INTEGER` that controls the number of products with A (and with A^T in the SVD case). If `iters`$<$`0`, no initial products with A are performed; this choice is suitable if the eigenvalues/singular values of A decay rapidly. If `iters`$=$`0`, AG is performed. If `iters`$\geq$`1`, then $r =$`iters` steps of the power method are performed; see the algorithm outlines in Section 4 for more details. The default value is `0`.

`fa14_value` is a scalar of type `INTEGER` that controls the use of the pseudo random number generator `hsl_fa14`. If it is set to a positive value, then this is used to reset the value of the generator word used by `hsl_fa14`. The default value is `0`.

`rand_method` is a scalar of type `INTEGER` that controls the generation of the entries of the random matrix G. Possible values are:

`0` uniform distribution on $(0,1)$.

`1` uniform distribution on $(-1,1)$.

`2` normal distribution, with mean 0 and standard deviation 1.

The default value is `2`.

2.4.2 The derived data type for holding information

The derived data type `mc81_info` is used to hold information. The components are:

`flag` is a scalar of type `INTEGER` that gives the exit status of the algorithm (details in Section 2.5).

`stat` is a scalar of type `INTEGER` that, in the event of an allocation error, holds the Fortran `stat` parameter (and is set to `0` otherwise).

2.5 Warning and error messages

A successful return is indicated by `info%flag` having the value zero. A negative value is associated with error as follows:

-1 Error in a scalar input parameter.

-2 Error in the size of array of a rank-two array (that is, `b`, `g`, `q`, `u`, `vt`, `w`, or `x`).

-3 Allocation error. The stat parameter is returned in `info%stat`.

-4 Unexpected error returned by LAPACK routine `_syev`.

-5 Unexpected error returned by LAPACK routine `_gesvd`.

-6 Error returned by LAPACK routine `_potrf`. This may occur if `algorithm` is set to `2` and A is not positive definite.

-7 One or more columns of A does not have the diagonal as the first entry (`MC81_revd` only).

-8 `ptr` is no monotonic increasing (`MC81_rsvd` only).

3 GENERAL INFORMATION

Other routines called directly: `HSL_FA14`, `KB08`, BLAS subroutines `_gemm`, `_gemv`, `_trsm`, and the LAPACK subroutines `_gesvd`, `_gemqrt`, `_geqrt`, `_potrf`, `_syev`.

Restrictions: `m` ≥ 0: `n`$\geq$`0`; `k`$>$`0`; `p`$\geq$`0`; `k+p`$<$`n`.

4 METHOD

Each routine starts by performing simple checks on the user-supplied data. If an error is found, an error flag is set and control is returned to the user. Note that only scalar arguments are checked: there are no checks on the matrix A for out-of-range or duplicated entries that could cause the package to fail in an unpredictable way. The following algorithms outline the methods that are then implemented within `HSL_MC81`. Here $r =$ `control%iters`. If the reverse communication variants are used, controlled is returned to the user each time a matrix-vector product involving A or A^T is required.

Algorithm 1 Eigenvalue decomposition: Randomised eigenvalue decomposition (REVD) and Nyström's method.
Input: $n \times n$ symmetric positive definite matrix A, a target rank $k > 0$, an over-sampling parameter $p \geq 0$, the number r of steps in the power iteration and the parameter `algorithm` that determines which method is used.
Output: Orthonormal matrix X of k approximate eigenvectors and a real diagonal matrix D of k approximate eigenvalues such that $A \approx XDX^T$.

```
1: Unless supplied by the user, form an $n \times (k+p)$ random matrix $G$.
2: if $r \geq 0$ then
3:     Form the $n \times (k+p)$ sample matrix $Y = AG$.
4:     Orthonormalize the columns of the sample matrix, that is, $Q = orth(Y)$ using _geqrt.
5:     for $j = 1, r$ do
6:         Compute $Q = orth(AQ)$.
7:     end for
8: else
9:     Orthonormalize the columns of the random matrix, that is, $Q = orth(G)$ using _geqrt.
10: end if
11: Form the matrices $B_1 = AQ$ and $B_2 = Q^T B_1$.
12: if algorithm $= 2$ then
13:     Perform a Cholesky factorization $B_2 = C^T C$ using _potrf.
14:     Solve $F = B_1 C^{-1}$ using _trsm.
15:     Compute an SVD $F = X \Sigma V^T$ using _gesvd and set $D = \Sigma^2$. Retain largest $k$ entries in $D$ and corresponding
    columns of $X$.
16: else
17:     Form the EVD $B_2 = UDU^T$ using _syev. Retain largest $k$ entries in $D$ and corresponding columns of $U$.
18:     Set $X = QU$.
19: end if
```

Algorithm 1 with the parameter `algorithm` $\neq 2$ is taken from Algorithms 4.4 and 5.3 of [1] and `algorithm` $= 2$ is taken from Figure 6 of [2]. If $r < 0$ then the latter is a single-pass algorithm (that is, it accesses A only once). Increasing r increases the accuracy but at additional cost. Algorithm 2 is taken from Algorithms 4.4 and 5.1 of [1].

References:

[1] N. Halko, P. G. Martinsson and J. A. Tropp (2011). Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review Vol. 53, No. 2, pp.217–288.

[2] Per-Gunnar Martinsson (2018). Randomized methods for matrix computations. In *The Mathematics of Data* (M. W. Mahoney, J. Duchi and A. Gilbert, eds), IAS/Park City Mathematics Series Vol. 25, pp 187–229

Algorithm 2 Randomised singular value decomposition (RSVD).
Input: $m \times n$ matrix A, a target rank $k > 0$, an over-sampling parameter $p \geq 0$ and the number $r \geq 0$ of steps in the power iteration.
Output: Unitary matrices U and V of k approximate left and right singular vectors and a diagonal matrix Σ of k approximate singular values such that $A \approx U\Sigma V^T$.

```
 1: if $r \geq 0$ then
 2:     Unless supplied by the user, form an $n \times (k+p)$ random matrix $G$.
 3:     Form the $m \times (k+p)$ sample matrix $Y = AG$.
 4:     Orthonormalize the columns of the sample matrix, that is, $Q = orth(Y)$ using _geqrt.
 5:     for $j = 1, r$ do
 6:         Compute $W = orth(A^T Q)$.
 7:         Compute $Q = orth(AW)$.
 8:     end for
 9: else
10:     Unless supplied by the user, form an $m \times (k+p)$ random matrix $G$.
11:     Orthonormalize the columns of the random matrix, that is, $Q = orth(G)$ using _geqrt.
12: end if
13: Form the $(k+p) \times n$ matrix $B = Q^T A$.
14: Form the SVD $B = \hat{U}\Sigma V^T$ using _gesvd. Retain largest $k$ entries in $D$ and corresponding columns of $\hat{U}$ and $V^T$.
15: Set $U = Q\hat{U}(1:k)$.
```

5 EXAMPLE OF USE

Consider the 4×4 real symmetric matrix.

$$A = \begin{pmatrix} 2.1 & & -0.23 & \\ & 1.7 & & 0.66 \\ -0.23 & & 0.30 & \\ & 0.66 & & 0.77 \end{pmatrix}.$$

The following code demonstrates the use of both the standard and reverse communication interfaces for approximating the largest eigenvalue and corresponding eigenvector of A.

```
! Simple code to illustrate use of HSL_MC81
   program main_test81
      use hsl_mc81_double
      implicit none
      integer, parameter :: wp   = kind(0d0)

      integer :: action ! must be set to 0 prior to first call
        ! 0 : terminated
        ! 1 : compute x(1:n,k+p+1:2*(k+p)) = A*x(1:n,1:k+p)
      integer :: n, ne
      integer :: k ! required rank
      integer :: p ! oversampling parameter
      real(wp), allocatable :: d(:) ! On exit, holds computed Ritz values
      real(wp), allocatable :: x(:,:) ! must be size at least n by 2(k+p)
         ! Used for reverse communication and on final exit x(1:n,1:k+p) holds
         ! computed Ritz vectors (approximate eigenvectors).
! Matrix A must be held in CSC format (lower triangle)
integer, allocatable :: ptr(:) ! column pointers
integer, allocatable :: row(:) ! matrix entries
real(wp), allocatable :: val(:) ! matrix entries

type(MC81_keep) :: keep
type(MC81_info) :: info
type(MC81_control) :: control

integer :: kp

!!!!!
read (5,*) n, ne, k, p
kp = k + p

! allocate arrays and then read matrix
allocate (d(k+p), ptr(n+1), row(ne), val(ne), x(n,2*(k+p)))

read (5,*) ptr(1:n+1)
read (5,*) row(1:ne)
read (5,*) val(1:ne)
!!!!!
control%iters = 2

! first run using the standard interface
write (*,'(/a)') ' Standard interface with algorithm = 1: '

call MC81_revd (n, ptr, row, val, k, p, d, x, control, info)
if (info%flag.lt.0) then
   call mc81_print_message(info, 6,' MC81_revd')
else
   write (*,'(a,es13.3)')    &
   ' computed approximate largest eigenvalue is: ',d(1)
end if

! Now rerun to demonstrate the reverse communication interface
action = 0

write (*,'(/a)') ' Reverse communication interface with algorithm = 2: '
do
   call MC81_revd_reverse (action, n, k, p, d, x, keep, &
        control, info, algorithm=2)

   if (action.eq.0) then
      if (info%flag.lt.0) then
         call mc81_print_message(info, 6,' MC81_revd_reverse')
      else
         write (*,'(a,es13.3)')               &
```

```
              ' computed approximate largest eigenvalue is: ',d(1)
            end if
            exit
         else if (action.eq.1) then
           ! compute x(1:n,kp:2*kp) = A*x(1:n,1:kp)
           call sym_mxmult (n, ne, ptr, row, val, kp, &
                x(:,1:kp), n, x(:,kp+1:2*kp),n)
        end if
     end do
contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   ! Simple routine to multiply symmetric A held in CSC format by x(:,1:r)
   ! and place result in y(:,1:r). Lower triangular part of A supplied,
   ! with the diagonal entry the first entry in each column.

   subroutine sym_mxmult(n,ne,ptr,row,val,r,x,ldx,y,ldy)

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

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

      do i = 1,n
        y(i,1:r) = x(i,1:r) * val(ptr(i))
      end do

     do j = 1, n
        ! loop over off diagonal entries
        do k = ptr(j)+1, ptr(j+1)-1
           i = row(k)
           temp = val(k)
           y(i,1:r) = y(i,1:r) + x(j,1:r)*temp
           y(j,1:r) = y(j,1:r) + x(i,1:r)*temp
        end do
     end do
   end subroutine sym_mxmult

   end program main_test81
```

With the input data:

```
   4   6   1   2
   1   3   5   6   7
   1   3   2   4   3   4
   2.1  -0.23  1.7  0.66  0.30  0.77
```

this produces the output

```
Standard interface with algorithm = 1:
computed approximate largest eigenvalue is:     2.129E+00

Reverse communication interface with algorithm = 2:
computed approximate largest eigenvalue is:     2.129E+00
```