Science and Technology Facilities Council

HSL_MC64

PACKAGE SPECIFICATION HSL

1 SUMMARY

Given a sparse matrix $A = \{a_{ij}\}_{m\times n}$, $m \geq n$, HSL_MC64 attempts to **find row and column permutation vectors** that make the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns permutations that maximize the smallest element on the diagonal, maximize the sum of the diagonal entries, or maximize the product of the diagonal entries of the permuted matrix. For the latter option, the subroutine also **finds scaling factors** that may be used to scale the original matrix so that the nonzero diagonal entries of the permuted and scaled matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. The natural logarithms of the scaling factors u_i, $i = 1, ..., m$, for the rows and v_j, $j = 1, ..., n$, for the columns are returned so that the scaled matrix $B = \{b_{ij}\}_{n\times n}$ has entries

$$b_{ij} = a_{ij}\, exp(u_i + v_j).$$

In this Fortran 95 version, there are added facilities from the original MC64 code for working on rectangular and symmetric matrices. We use the term *matching* to mean a set of entries in the matrix no two of which are in the same row or column. For the rectangular case, the user can permute the matching to the diagonal and identify the rows in the structurally nonsingular block. The matching will lie on the diagonal of the permuted matrix. For the symmetric case, the user must only supply the lower triangle and, if a scaling is computed, it will be a symmetric scaling with the same property as in the unsymmetric case. In this case, a symmetric permutation is returned but the matching will not normally be on the diagonal of the permuted matrix. If the matrix is structurally singular and symmetric the user may call the maximum product matching only. In this case the matching returned will not be complete, but is optimal on a submatrix. The scaling will be such that the same property holds (i.e. all entries are less than one in absolute value).

ATTRIBUTES — Version: 2.4.3 (1 November 2023) **Interfaces:** C, Fortran, MATLAB. **Types:** Real (single,double), Complex (single,double). **Calls:** HSL_MC34, MC64, MF64, HSL_ZD11. **Original date:** July 2007. Version 2.0.0 May 2009. **Origin:** I. S. Duff, Rutherford Appleton Laboratory, and J. Koster, Trondheim. Also J. D. Hogg, Rutherford Appleton Laboratory (Version 2.1.0 and later). **Language:** Fortran 95. **Remark:** HSL_MC64 is a Fortran 95 encapsulation of MC64 and offers additional facilities to the Fortran 77 version. The work of the first author was supported by EPSRC Grant EP/E053351/1.

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_MC64_SINGLE
```

Double precision version
```
USE HSL_MC64_DOUBLE
```

Single precision complex version
```
USE HSL_MC64_COMPLEX
```

Double precision complex version
```
USE HSL_MC64_DOUBLE_COMPLEX
```

If it is required to use multiple modules at the same time, the derived types MC64_CONTROL (Section 2.8) and MC64_INFO (Section 2.9) must be renamed in one of the USE statements.

The following subroutines are available to the user:

(a) Use of MC64_INITIALIZE is optional. As of version 2.2.0 it is **deprecated** as components of MC64_CONTROL assume default values upon instantiation. MC64_INITIALIZE may be called to set default values for the components of the control structure. If non-default values are wanted for any of the control components, the corresponding components should be altered after the call to MC64_INITIALIZE.

(b) MC64_MATCHING accepts the matrix A and finds the required permutations (and possibly a scaling).

2.2 Input of the matrix

2.2.1 HSL standard format (v2.2.0 and later)

The user should supply the matrix A in standard HSL format. If A is symmetric, only the **lower** triangular part should be supplied. HSL standard format is compressed sparse column format with the entries within each column ordered by increasing row index. There is no requirement that zero entries on the diagonal are explicitly included. Before using HSL_MC64, 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 standard HSL format. The input of A is illustrated in Section 5.

2.3 HSL_ZD11 derived data type (deprecated)

The derived data type ZD11_TYPE may be used to hold the sparse matrix, and is supported for backwards compatibility. As of version 2.2.0, the use of HSL standard format is preferred. The following components are employed:

M is an INTEGER scalar that holds the number of rows m in the matrix A. **Restriction:** M $\geq$ N.

N is an INTEGER scalar that holds the number of columns n in the matrix A. **Restriction:** N ≥ 1.

NE is an INTEGER scalar that holds the number of matrix entries. **Restriction:** NE ≥ 0.

VAL is a package type allocatable rank-1 array of size at least NE, the leading part of which holds the values of the entries stored by columns.

ROW is an INTEGER allocatable rank-1 array of size at least NE, the leading part of which holds the row indices of the entries so that entry VAL(k) is in row ROW(k), $k = 1, \ldots$NE.

PTR is an INTEGER allocatable rank-1 array of size at least N+1, the leading part of which holds the position of the start of each column in the matrix ordered by columns in VAL and ROW. PTR($n+1$) is equal to NE+1.

ID is a CHARACTER allocatable rank-1 array of size at least 1. If the user is inputting the lower triangle of a symmetric matrix (not all diagonal entries need to be present), MATRIX%ID should be set to 'S'.

The other components of the type are not used.

2.4 Argument lists and calling sequences

2.4.1 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 all optional arguments be called by keyword, not by position.**

2.4.2 Package types

REAL denotes default real if the single precision version or the complex version is being used, and double precision real if the double precision or double precision complex version is being used. We use the term **package type** to mean default real if the single precision version is being used, double precision real for the double precision version, default complex for the complex version and double precision complex for the double complex version.

2.4.3 The initialization subroutine (deprecated)

The initialization subroutine may be called to set default values for the components of the derived type MC64_CONTROL. The use of this subroutine is deprecated in version 2.2.0 as default values are now assigned in the definition of the type. If the used, any changes to parameter values, should be performed after the call to MC64_INITIALIZE.

```
CALL MC64_INITIALIZE(CONTROL)
```

CONTROL is a scalar INTENT(OUT) argument of type MC64_CONTROL_TYPE (see Section 2.8). On exit, CONTROL contains default values for the components as described in Section 2.8.

2.5 To find the column permutation (and possibly a scaling)

As of Version 2.2.0, an interface using HSL standard format is defined. The old interface using HSL_ZD11 is still supported, but is deprecated and may be removed at a later date.

```
CALL MC64_MATCHING(JOB,MATRIX_TYPE,M,N,PTR,ROW,VAL,CONTROL,INFO,PERM[,SCALE])   (preferred)
CALL MC64_MATCHING(JOB,MATRIX,CONTROL,INFO,PERM[,SCALE])                       (deprecated)
```

JOB is an INTEGER scalar with INTENT(IN). On entry, it must be set by the user to control the action. It is not altered by the subroutine. Possible values for JOB are:

1 Compute a row and column permutation of the matrix so that the permuted matrix has as many entries on its diagonal as possible. The values on the diagonal are of arbitrary size.

2 Compute a row and column permutation of the matrix so that the smallest value on the diagonal of the permuted matrix is maximized.

3 Compute a row and column permutation of the matrix so that the smallest value on the diagonal of the permuted matrix is maximized. The algorithm differs from the one used for JOB=2 and may have quite a different performance (see Section 4).

4 Compute a row and column permutation of the matrix so that the sum of the diagonal entries of the permuted matrix is maximized.

5 Compute a row and column permutation of the matrix so that the product of the diagonal entries of the permuted matrix is maximized. Vectors are also computed to scale the matrix so that the nonzero diagonal entries of the permuted matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. In the symmetric case the scaling will be symmeterised as described in Section 4.

MATRIX_TYPE is an INTEGER scalar that indicates if the matrix is symmetric or not. Values of 3 or greater are treated as symmetric, all other values are treated as unsymmetric/rectangular depending on the values of M and N (this is consistent with the matrix types defined by the HSL_MC69 package).

M is an INTEGER scalar that holds the number of rows m in the matrix A. **Restriction:** M $\geq$ N.

N is an INTEGER scalar that holds the number of columns n in the matrix A. **Restriction:** N $\geq$ 1.

PTR is an INTEGER rank-1 array of size at least 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.

ROW is an INTEGER rank-1 array of size at least PTR(N+1)-1. It must hold the row indices of the entries of A with those for column 1 preceding those for column 2, and so on (within each column, the row indices may be in arbitrary order). If A is symmetric, only those entries in the lower triangle should be stored.

VAL is a package type rank-1 array of size at least PTR(N+1)-1, the leading part of which holds the values of the entries stored by columns such that VAL(k) is the value of the entry in ROW(k).

MATRIX is a scalar INTENT(IN) argument of type ZD11_TYPE. The user must set the components M, N, NE, ROW, and PTR, and they are not altered by the subroutine. **Restriction:** MATRIX%M $\geq$ MATRIX%N, MATRIX%N $\geq$ 1, and MATRIX%NE $\geq$ 0.

CONTROL is a scalar INTENT(IN) argument of type MC64_CONTROL_TYPE (see Section 2.8). Default values may be assigned by calling MC64_INITIALIZE prior to the first call to MC64_MATCHING.

INFO is a scalar INTENT(OUT) argument of type MC64_INFO_TYPE (see Section 2.9). A successful call to MC64_MATCHING is indicated when the component FLAG has the value 0.. For other return values of FLAG, see Section 2.6.

PERM is an rank-1 array of size (M+N) of INTENT(OUT) and of type INTEGER. ABS(PERM(i)), $i = 1,2,\ldots,$M+N, will be set to permutation vectors. The entries i $= 1,2,\ldots,$M hold a permutation vector such that row i of the original matrix is row ABS(PERM(i)) in the permuted matrix. Column j of the original matrix is column ABS(PERM(M+j)) in the permuted matrix. For rows and columns that are unmatched, the value of PERM is negative.

SCALE is an optional rank-1 array of size (M+N) of INTENT(OUT) and of type REAL. If present and JOB $= 5$, SCALE(i), i $= 1,2,...,$M, and SCALE(M+j), j $= 1,2,...,$N will be set to the row and column scaling vectors u_i and v_j as discussed in Section 1. Note that these scale factors are applied to the original (unpermuted) matrix. If JOB $\neq 5$, scale is not accessed.

2.6 Warning and error messages

A negative value of INFO%FLAG on exit from MC64_MATCHING indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

- 1 Value of JOB out of range. JOB > 5 or JOB ≤ 0. INFO%MORE is set to the value of JOB. Note that further options for JOB may be available in later releases.

- 2 Value of N out of range. N < 1. INFO%MORE is set to the value of N.
(Value of MATRIX%N out of range. MATRIX%N < 1. INFO%MORE is set to the value of MATRIX%N.)

- 3 Value of PTR(N+1)-1 out of range (PTR(N+1)-1 < 1). INFO%MORE is set to the value of PTR(N+1)-1.
(Value of MATRIX%PTR(MATRIX%N+1)-1 out of range (MATRIX%PTR(MATRIX%N+1)-1 < 1) or MATRIX%PTR(MATRIX%N+1)-1 $\neq$ MATRIX%NE. INFO%MORE is set to the value of MATRIX%PTR(MATRIX%N+1)-1.)

- 4 Value of M $<$ N. INFO%MORE is set to the value of M.
(Value of MATRIX%M $<$ MATRIX%N. INFO%MORE is set to the value of MATRIX%M.)

- 5 Failure of an allocate or deallocate statement. INFO%STAT is set to the STAT value. INFO%MORE is set to the value of N.

- 6 Index out of range. INFO%MORE is set to the position in array ROW where out of range index found.
(Index out of range. INFO%MORE is set to the position in array MATRIX%ROW where out of range index found.)

- 7 Duplicate entry. INFO%MORE is set to the position in the array ROW where duplicate found.
(Duplicate entry. INFO%MORE is set to the position in the array MATRIX%ROW where duplicate found.)

- 8 A symmetric matrix was indicated but there were entries in the upper triangle in column INFO%MORE.

A positive value of INFO%FLAG on exit from MC64_MATCHING is used to warn the user that the data may be faulty or that the subroutine cannot guarantee the solution obtained. Possible values are:

+ 1 Matrix is structurally singular. The cardinality of the matching is less than N.

+ 2 The returned scaling factors are large and may cause overflow when used to scale the matrix (JOB=5, only).

2.7 The derived data types

Two derived data types are accessible from the package. For each problem, the user must employ derived types defined by the module to declare structures for controlling the operation of the routine and returning information.

2.8 The control derived data type

The derived data type MC64_control is used to control the action. The components, which are automatically given default values in the definition of the type (and can be reset to the default values by a call to MC64_INITIALIZE), are:

LP is an INTEGER scalar that is used as the output stream for error messages. If it is negative, these messages will be suppressed. The default value is 6.

WP is an INTEGER scalar that is used as the output stream for warning messages. If it is negative, these messages will be suppressed. The default value is 6.

SP is an INTEGER scalar that is that is used as the output stream for diagnostic messages. If it is negative, these messages will be suppressed. The default value is -1.

LDIAG is a scalar variable of type default INTEGER that is used to control the amount of informational output that is required. No informational output will occur if LDIAG $\leq$ 0. The levels are:

<1 No messages are output.

1 Only error messages are output.

2 Error and warning messages output.

3 As for 2 plus scalar parameters, the first 10 entries of array parameters, and the control parameters on the first entry to MC64_MATCHING.

>3 All data will be printed on entry and exit.

The default is LDIAG = 2.

CHECKING is a scalar variable of type default INTEGERthat is used by MC64_MATCHING to specify whether the input data is checked for out-of-range indices and duplicates. This will be done if CHECKING is set to zero. These checks will not be performed if CHECKING is set to any other value. MC64_MATCHING will run faster without the checking but, if either out-of-range indices or duplicates are present, it may fail so the user must be sure of the data before changing the default. If the user has already performed checking or conversion using HSL_MC69 this option is redundant and should be disabled. The default value is 0.

2.9 The derived data type for holding information

The derived data type MC64_INFO is used to hold information from the execution of MC64_MATCHING. The components are:

FLAG is an INTEGER scalar used as an error/warning flag. Negative values indicate a fatal error and positive values are associated with a warning. Details in Section 2.6.

MORE is an INTEGER scalar that provides further information in the case of an error, see Section 2.6.

STRUCRANK is an INTEGER scalar that gives the number of entries that can be placed on the diagonal (the structural rank).

STAT is a scalar of type INTEGER. In the case of the failure of an allocate or deallocate statement, it is set to the value of the Fortran stat parameter.

3 GENERAL INFORMATION

Input/output: Output is provided under the control of CONTROL%LDIAG. If a non-negative value is set in CONTROL%LP, CONTROL%WP, or CONTROL%SP, then error, warning or diagnostic information will be printed to the corresponding unit number, respectively. However if a value is negative, printing is suppressed.

Restrictions: M≥N, N≥1. (MATRIX%M≥MATRIX%N, MATRIX%N≥1 and MATRIX%NE≥0.)

Changes from Version 1.0.0 The main change from Version 1.0.0 is that both a row and column permutation are provided in all cases (symmetric, unsymmetric, and rectangular) so that the permuted matrix is $A(permi, permj)$ where *permi* is the row permutation and *permj* is the column permutation. In the symmetric case, *permi* = *permj*, in the rectangular case *permj* is the identity, and in the unsymmetric case *permi* is the identity. The error flags have also been changed so that they are the same as the Fortran 77 package MC64 for the same error.

Changes from Version 2.1.0 A new interface using HSL standard format was added; the old interface using HSL_ZD11 was marked deprecated. The MC64_CONTROL structure was given default initialisation values; the subroutine MC64_INITIALIZE was marked deprecated. Behaviour for singular symmetric matrices with JOB = 5 improved.

4 METHOD

The algorithms used in the code and a discussion of its performance are given in detail in the paper [4], and we give only a very brief summary here. Earlier work on these codes and a study of their use in solving systems by both iterative and direct methods is given in [3].

The algorithm that is used for JOB = 1 is MC21, a depth-first search algorithm with look-ahead technique [2].

The algorithm that is used for JOB = 2 solves a bottleneck bipartite matching problem. The algorithm starts with a partial matching and extends this by repeatedly searching the bipartite graph corresponding to the matrix for a path

from an unmatched column node to any unmatched row node and for which the smallest weight of any edge in this path is maximal.

The algorithm for JOB = 3 is based on repeated applications of an MC21 type algorithm to matrices A_e obtained from the original $m \times n$ matrix A by deleting those entries from A that are below a certain threshold value e. If a column permutation of cardinality n exists for A_e, the threshold value e is increased; otherwise, e is decreased. This is repeated until e is such that a maximum matching of size n exists for matrix A_e, but not for $A_{e'}$, where $e' > e$. This algorithm is described in detail in [3].

The algorithm that is used for JOB = 4 and 5 solves a weighted bipartite matching problem. The algorithm starts with a partial matching and extends this by repeatedly searching the corresponding bipartite graph for a path from an unmatched column node to any unmatched row node whose length is shortest. These paths are found using an algorithm similar to that of Dijkstra [1].

If the input matrix is symmetric, then the matrix is first expanded using MC34 before calling MC64. A symmetric permutation will be returned which means that any matching entries that are off-diagonal in the original matrix will not be permuted to the diagonal. The scaling, if requested, is set to the square root of the product of the row and column scaling computed by MC64 and will maintain the property that there is at least one entry in every row and column of the scaled matrix with absolute value one and all other entries have modulus less than or equal to one.

If the matrix is symmetric, structurally singular and JOB = 5, a partial matching representing a non-singular submatrix of maximum rank will be returned. The scaling, if requested, is set as for the non-singular case on this submatrix. For unmatched rows and columns, it is set such that

$$s_i = \log \frac{1}{\max_{k \in index(\bar{A})} |a_{ik} s_k|} \qquad \text{with the convention } \frac{1}{0} = 1.$$

where $\bar{A}$ is the non-singular submatrix of maximum rank.

If the matrix is rectangular, the permuted matrix will have matching entries in the n positions

$$a(permi(i), permc(i)), i = 1, ...n$$

and the scaled matrix will have at least one entry of modulus one in each column and row with no entries of modulus greater than one.

References

[1] Dijkstra, E. W. A note on two problems in connection with graphs. Numerische Mathematik **1**, 269-271, 1959.

[2] Duff, I. S. Algorithm 575. Permutations for a zero-free diagonal. ACM Trans Math Softw. **7**, 387-390, 1981.

[3] Duff, I. S. and Koster, J. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM J Matrix Analysis and Applications **20** (4), 889-901, 1999.

[4] Duff, I. S. and Koster, J. On algorithms for permuting large entries to the diagonal of a sparse matrix. SIAM J Matrix Analysis and Applications **22** (4), 973-996, 2001.

5 EXAMPLE OF USE

The following program reads sparse matrices by columns and calls HSL_MC64 to compute a permutation and scaling. It prints the permutation, the scaling and the scaled matrix.

```
PROGRAM HSL_MC64DS
   USE HSL_MC64_DOUBLE
   IMPLICIT NONE
   TYPE(MC64_CONTROL) CONTROL
   TYPE(MC64_INFO) INFO

   INTEGER :: MATRIX_TYPE
   INTEGER, ALLOCATABLE :: PTR(:), ROW(:)
   DOUBLE PRECISION, ALLOCATABLE :: VAL(:)

   INTEGER, ALLOCATABLE :: PERM(:)
   INTEGER EYE, I, J, JAY, JOB, K, KASE, M, N, NE

   INTEGER, PARAMETER :: WP = KIND(0.0D0)

   REAL(WP), ALLOCATABLE :: SCALE(:),A(:,:)

   REAL(WP), PARAMETER :: ZERO=0.0D0

   DO KASE = 1,3
! KASE = 1 ... square matrix
! KASE = 2 ... rectangular matrix
! KASE = 3 ... symmetric matrix
      IF (KASE == 1) THEN
         WRITE(6,'(A)') 'Square matrix'
         MATRIX_TYPE = 2
      ENDIF
      IF (KASE == 2) THEN
         WRITE(6,'(//A)') 'Rectangular matrix'
         MATRIX_TYPE = 1
      ENDIF
      IF (KASE == 3) THEN
         WRITE(6,'(//A)') 'Symmetric matrix'
         MATRIX_TYPE = 4
      ENDIF

! Read matrix order and number of entries
      READ (5,*) M,N,NE

! Allocate arrays of appropriate sizes
      ALLOCATE(PTR(N+1), VAL(NE), ROW(NE))
      ALLOCATE(PERM(M+N),SCALE(M+N))
      ALLOCATE(A(M,N))

! Read matrix and right-hand side
      READ (5,*) (PTR(I),I=1,N+1)
      READ (5,*) (ROW(I),I=1,NE)
      READ (5,*) (VAL(I),I=1,NE)
```

```
! Get matching
      JOB = 5
      CALL MC64_MATCHING(JOB,MATRIX_TYPE,M,N,PTR,ROW,VAL,CONTROL,INFO,PERM,SCALE)
      IF(INFO%FLAG<0) THEN
         WRITE(6,'(A,I2)') &
            ' Failure of MC64_MATCHING with INFO%FLAG=', INFO%FLAG
         STOP
      END IF

! Print permutations
      WRITE(6,'(A/(10I8))') 'Row permutation',PERM(1:M)
      WRITE(6,'(A/(10I8))') 'Column permutation',PERM(M+1:M+N)
      IF (MATRIX_TYPE.EQ.4) THEN
! Print symmetric scaling
        WRITE(6,'(A/(5D12.4))') 'Symmetric scaling',EXP(SCALE(1:N))
      ELSE
! Print row scaling
        WRITE(6,'(A/(5D12.4))') 'Row scaling',EXP(SCALE(1:M))
! Print column scaling
        WRITE(6,'(A/(5D12.4))') 'Column scaling',EXP(SCALE(M+1:M+N))
      ENDIF

! Scale matrix and put scaled and permuted entries in full array
      A(1:M,1:N) = ZERO
      DO J = 1,N
! EYE and JAY are indices in permuted matrix
        JAY = ABS(PERM(M+J))
        DO K = PTR(J), PTR(J+1)-1
          I = ROW(K)
          EYE = ABS(PERM(I))
! Scale and permute the matrix
          A(EYE,JAY) = EXP(SCALE(I))*VAL(K)*EXP(SCALE(M+J))
! Set symmetric counterpart if appropriate
          If (MATRIX_TYPE.GE.3) A(JAY,EYE) = A(EYE,JAY)
        ENDDO
      ENDDO

! Write out scaled matrix
      WRITE(6,'(A)') 'Scaled matrix'
      DO I = 1,M
        WRITE(6,'(3D12.4)') (A(I,J),J=1,N)
      ENDDO

    ENDDO
END PROGRAM HSL_MC64DS
```

with the following data

```
3 3 5
1 3 5 6
```

```
2 3 2 3 1
8.0 3.0 2.0 1.0 4.0
3 2 6
1 4 7
1 2 3 1 2 3
5.0 2.0 1.0 1.0 3.0 4.0
3 3 4
1 2 4 5
2 2 3 3
2.0 1.0 1.0 4.0
```

This produces the following output:

```
Square matrix
Row permutation
       1       2       3
Column permutation
       2       3       1
Row scaling
  0.1000D+01  0.1000D+01  0.2000D+01
Column scaling
  0.1250D+00  0.5000D+00  0.2500D+00
Scaled matrix
  0.1000D+01  0.0000D+00  0.0000D+00
  0.0000D+00  0.1000D+01  0.1000D+01
  0.0000D+00  0.7500D+00  0.1000D+01


Rectangular matrix
Row permutation
       1       2      -3
Column permutation
       1       2
Row scaling
  0.1000D+01  0.1333D+01  0.1000D+01
Column scaling
  0.2000D+00  0.2500D+00
Scaled matrix
  0.1000D+01  0.2500D+00
  0.5333D+00  0.1000D+01
  0.2000D+00  0.1000D+01


Symmetric matrix
Row permutation
       2       1       3
Column permutation
       2       1       3
Symmetric scaling
  0.7071D+00  0.7071D+00  0.5000D+00
```

```
Scaled matrix
  0.5000D+00  0.1000D+01  0.3536D+00
  0.1000D+01  0.0000D+00  0.0000D+00
  0.3536D+00  0.0000D+00  0.1000D+01
```