1 SUMMARY

Given a sparse complex matrix $\mathbf{A} = \{a_{ij}\}_{n\times n}$, this subroutine attempts to **find a column permutation vector** that makes the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns a column permutation that maximizes the smallest modulus of an entry on the diagonal, maximizes the sum of the moduli of the diagonal entries, or maximizes the product of the moduli 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 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, $i = 1, ..., n$, for the rows and v_j, $j = 1, ..., n$, for the columns are returned so that the scaled matrix $\mathbf{B} = \{b_{ij}\}_{n\times n}$ has entries

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

ATTRIBUTES — **Version:** 1.1.0. (26 March 2013) **Types:** Real (single, double). **Calls:** `MC21`, `MC64`. **Original date:** August 2007. **Origin:** I. S. Duff and J. Koster (Rutherford Appleton Laboratory). **Language:** Fortran 77.

2 HOW TO USE THE PACKAGE

2.1.1 To set default values of controlling parameters

If non-default values for any of the control parameters are required, they should be set immediately after the call to `MF64I/ID`.

The single precision version

```
CALL MF64I(ICNTL)
```

The double precision version

```
CALL MF64ID(ICNTL)
```

`ICNTL` is an `INTEGER` array of length `10` that need not be set by the user. On return it contains default values.

`ICNTL(1)` specifies the output stream for error messages. If `ICNTL(1)` $<$ `0`, these messages are suppressed. The default value, set by `MF64I/ID`, is `6`.

`ICNTL(2)` specifies the output stream for warning messages. If `ICNTL(2)` $<$ `0`, these messages are suppressed. The default value, set by `MF64I/ID`, is `6`.

`ICNTL(3)` specifies the output stream for diagnostic messages. If `ICNTL(3)` $<$ `0`, these messages are suppressed. The default value, set by `MF64I/ID`, is `-1`.

`ICNTL(4)` specifies whether the input data is checked for out-of-range indices and duplicates. The default value of `0`, set by `MF64I/ID`, will invoke data checking. These checks will not be performed if `ICNTL(4)` is set to any other value. `MF64A/AD` 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.

`ICNTL(5)` to `ICNTL(10)` are not currently used but are set to zero by `MF64I/ID`.

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

The single precision version

```
CALL MF64A(JOB,N,NE,IP,IRN,A,NUM,CPERM,LIW,IW,LDW,DW,ICNTL,INFO)
```

The double precision version

```
CALL MF64AD(JOB,N,NE,IP,IRN,A,NUM,CPERM,LIW,IW,LDW,DW,ICNTL,INFO)
```

`JOB` is an `INTEGER` variable that 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 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 column permutation of the matrix so that the smallest value on the diagonal of the permuted matrix is maximized.
3. Compute a 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 column permutation of the matrix so that the sum of the diagonal entries of the permuted matrix is maximized.
5. Compute a 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.

Restriction: $1 \leq$ `JOB` ≤ 5.

`N` is an `INTEGER` variable that must be set by the user to the order of the matrix. It is not altered by the subroutine. **Restriction:** `N` ≥ 1.

`NE` is an `INTEGER` variable that must be set by the user to the number of entries in the matrix. It is not altered by the subroutine. **Restriction:** `NE` ≥ 1.

`IP` is an `INTEGER` array of length `N+1`. The user must set `IP(J)` to contain the position in array `IRN` of the first row index of an entry in column `J`, for `J` = 1, ..., `N`. `IP(N+1)` must be set to `NE+1`. It is not altered by the subroutine.

`IRN` is an `INTEGER` array of length `NE`. The user must set `IRN` to hold the row indices of the entries of the matrix. Those belonging to column `J` must be stored contiguously in the positions `IP(J)`, ..., `IP(J+1)-1`. The ordering of the row indices within each column is unimportant. Repeated entries are not allowed. The array `IRN` is not altered by the subroutine.

`A` is a `COMPLEX` (`COMPLEX*16` in the D version) array of length `NE`. The user must set `A(K)` to the value of the entry that corresponds to `IRN(K)`, `K` = 1, ..., `NE`. It is not used by the subroutine when `JOB` = 1. It is not altered by the subroutine.

`NUM` is an `INTEGER` variable that need not be set by the user. On successful exit, `NUM` will be the number of entries on the diagonal of the permuted matrix. If `NUM` is less than `N`, the matrix is structurally singular. For an example of this, see Section 2.2.

`CPERM` is an `INTEGER` array of length `N` that need not be set by the user. On successful exit, `ABS(CPERM)` contains the column permutation. If the matrix is structurally singular (`NUM` < `N`), then the entries of `CPERM` for which there is no entry on the diagonal of the permuted matrix are set negative. Column `ABS(CPERM(J))` of the original matrix is column `J` in the permuted matrix, `J` = 1, ..., `N`.

`LIW` is an `INTEGER` variable that must be set by the user to the dimension of array `IW`. It is not altered by the subroutine.

Restriction:

JOB = 1 : LIW ≥ 5N.
JOB = 2 : LIW ≥ 4N.
JOB = 3 : LIW ≥ 10N+NE.
JOB = 4 : LIW ≥ 5N.
JOB = 5 : LIW ≥ 5N.

IW is an INTEGER array of length LIW that is used for workspace.

LDW is an INTEGER variable that must be set by the user to the dimension of array DW. It is not altered by the subroutine.

Restriction:

JOB = 1 : LDW is not used.
JOB = 2 : LDW ≥ N.
JOB = 3 : LDW ≥ NE.
JOB = 4 : LDW ≥ 2N+NE.
JOB = 5 : LDW ≥ 3N+NE.

DW is a REAL (DOUBLE PRECISION in the D version) array of length LDW that is used for workspace. It is not used by the subroutine when JOB = 1. If JOB = 5, on return, DW(I) contains u_I, I = 1, ..., N, and DW(N+J) contains v_J, J = 1, ..., N. The column scaling factors v_J apply to the original matrix, not to the permuted matrix.

ICNTL is an INTEGER array of length 10. Its components control the output of MF64A/AD and must be set by the user before calling MF64A/AD. Default values can be set by calling MF64I/ID. The components are defined as in Section 2.1.1. ICNTL(5) to ICNTL(10) are not currently accessed by MF64A/AD. ICNTL is not altered by the subroutine.

INFO is an INTEGER array of length 10 that need not be set by the user. INFO(1) is set non-negative to indicate success. A negative value is returned if an error occurred, a positive value if a warning occurred. If INFO(1) < 0, then further information on the error is given in INFO(2). On exit from the subroutine, INFO(1) will take one of the following values:

0 successful entry (for a structurally nonsingular matrix).

+1 successful entry (for a structurally singular matrix).

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

–1 JOB < 1 or JOB > 5. Value of JOB held in INFO(2).

–2 N < 1. Value of N held in INFO(2).

–3 NE < 1. Value of NE held in INFO(2).

–4 the defined length LIW violates the restriction on LIW. Value of LIW required given by INFO(2).

–5 the defined length LDW violates the restriction on LDW. Value of LDW required given by INFO(2).

–6 an entry is found whose row index is out of range. INFO(2) contains the index of a column in which such an entry is found.

–7 repeated entries are found. INFO(2) contains the index of a column in which such entries are found.

INFO(3) to INFO(10) are not currently used but are set to zero by MF64A/AD.

2.2 Warning return

The user may input a matrix that is structurally singular (for which there is no permutation that puts N entries onto the diagonal). An example of this is

$$\begin{pmatrix} \times & \times \\ 0 & 0 \end{pmatrix}$$

In such cases, MF64A/AD will return with a warning (INFO = 1) and a permutation that will put as many entries on the diagonal as possible (1 in the above example). The array ABS(CPERM) will still hold a permutation of the integers 1, ..., N, but N-NUM of the pairs (I,ABS(CPERM(I))) are not entries in the matrix and the corresponding entry of CPERM will be negative.

3 GENERAL INFORMATION

Other routines called directly: MC21A/AD, MC64A/AD. The subroutine internal to the package is MF64B/BD. This subroutine need never be called directly by the user.

Input/output: Output is under control of argument ICNTL.

Restrictions: N ≥ 1, NE ≥ 1, 1 ≤ JOB ≤ 5. See also arguments LIW and LDW.

4 METHOD

The algorithms used in the code and a discussion of its performance are given in detail in the report [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 MC21A/AD, 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 $n \times n$ 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].

This code for complex matrices is derived from the real version. This was quite a simple procedure since most of the subsidiary routines work with the modulus (or logarithm of the modulus) of the entries of the input matrix.

References

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

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

[3] Duff, I. S. and Koster, J. (1997). The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. and Applics.* **20**, no. 4, 889-901.

[4] Duff, I. S. and Koster, J. (1999). On algorithms for permuting large entries to the diagonal of sparse matrices. Report RAL-TR-1999-030, Rutherford Appleton Laboratory.

5 EXAMPLE OF USE

The following example reads a sparse matrix by columns and calls MF64AD to compute a column permutation for JOB = 1, ..., 5. It subsequently uses the output for JOB = 5 to scale and permute the matrix so as to put entries with absolute value one on the diagonal and entries with absolute value at most one off the diagonal.

```
       INTEGER MAXN, MAXNE, LIW, LDW
       PARAMETER( MAXN=100, MAXNE=MAXN*MAXN,
      &           LIW=10*MAXN+MAXNE, LDW=3*MAXN+MAXNE )
       INTEGER ICNTL(10), INFO(10), I, J, JAY, JOB, K, N, NE, NUM,
      &        IRN(MAXNE), LENC(MAXN), IP(MAXN), CPERM(MAXN), IW(LIW)
       COMPLEX*16 A(MAXNE)
       DOUBLE PRECISION  DW(LDW)
       EXTERNAL MF64ID, MF64AD
       INTRINSIC EXP

 C Set default values for ICNTL
       CALL MF64ID(ICNTL)

 C Read matrix and set column pointers
       READ(5,*) N, NE
       IP(1) = 1
       DO 10 J=1,N
         READ(5,'(I4,3(I6,2F6.2))') LENC(J),
      &            (IRN(K),A(K),K=IP(J),IP(J)+LENC(J)-1)
         IP(J+1) = IP(J) + LENC(J)
    10 CONTINUE

 C Compute matchings
       DO 20 JOB = 1,5
        CALL MF64AD(JOB,N,NE,IP,IRN,A,NUM,CPERM,LIW,IW,LDW,DW,ICNTL,INFO)
         IF (INFO(1).NE.0) THEN
           WRITE(6,'(A,I2)')
      &    ' Error or warning from MF64A/AD, INFO(1) = ',INFO(1)
         ELSE
           WRITE(6,'(A,I1,A,4I2)')
      &    ' For JOB = ',JOB,' the permutation is: ',(CPERM(J),J=1,N)
         ENDIF
    20 CONTINUE

 C DW(1:N) contains row scaling, DW(N+1:2N) contains column scaling
 C Scale the matrix
       DO 25 J = 1,N
         DO 24 K = IP(J),IP(J+1)-1
           I = IRN(K)
           A(K) = A(K) * EXP(DW(I) + DW(N+J))
    24   CONTINUE
    25 CONTINUE

 C Write scaled and permuted matrix (absolute values)
       WRITE(6,'(//2A/A/)') 'The absolute values of the ',
      &     'scaled and permuted matrix (JOB=5) are:',
      &     'Column length then row index/value for each entry in column'
       DO 51 JAY=1,N
         J = ABS(CPERM(JAY))
         WRITE(6,'(A,2I4,4(I6,F6.2))') 'Column ',JAY,LENC(J),
      &           (IRN(K),ABS(A(K)),K=IP(J),IP(J)+LENC(J)-1)
    51 CONTINUE

       STOP
       END
```

Thus for the matrix

$$\begin{pmatrix} 3.00+1.00i & 5.00+2.00i & & \\ 2.00+1.00i & 3.00+2.00i & & 0.00+0.00i \\ 3.00+1.00i & & 4.00+3.00i & 6.00+4.00i \\ & & 1.00+3.00i & 2.00+4.00i \end{pmatrix}$$

we would have as input

```
4  10
3     3  3.00  1.00     1  3.00  1.00     2  2.00  1.00
2     1  5.00  2.00     2  3.00  2.00
2     4  1.00  3.00     3  4.00  3.00
3     2  0.00  0.00     3  6.00  4.00     4  2.00  4.00
```

and we would get the following output

```
 For JOB = 1 the permutation is:  2 4 1 3
 For JOB = 2 the permutation is:  1 2 4 3
 For JOB = 3 the permutation is:  1 2 4 3
 For JOB = 4 the permutation is:  2 1 4 3
 For JOB = 5 the permutation is:  2 1 4 3

The absolute values of the scaled and permuted matrix (JOB=5) are:
Column length then row index/value for each entry in column

Column    1   2     1  1.00     2  0.95
Column    2   3     3  1.00     1  1.00     2  1.00
Column    3   3     2  0.00     3  1.00     4  0.98
Column    4   2     4  1.00     3  1.00
```

The last five lines of this output represent the matrix

$$\begin{pmatrix} 1.00 & 0.95 & & \\ 1.00 & 1.00 & & 1.00 \\ & 0.00 & 1.00 & 0.98 \\ & & 1.00 & 1.00 \end{pmatrix}.$$