1 SUMMARY

To **solve an unsymmetric banded system of linear equations.** Given a unsymmetric band matrix $\mathbf{A}=\{a_{ij}\}_{n\times n}$ and an n-vector $\mathbf{b}$, this subroutine solves the system $\mathbf{Ax}=\mathbf{b}$.

The matrix is factorized using Gaussian elimination with row interchanges. The matrix is stored with a fixed bandwidth form, but advantage is taken of any variability of bandwidth. If the lower semibandwidth is kl and the upper semibandwidth is ku, that is, if $a_{ij}=0$ for $i>j+kl$ or $j>i+ku$, fill-in is limited to kl additional diagonals of the upper triangle and the computation is performed within an array of size $n(2kl+ku+1)$.

At each pivotal step, operations are avoided on any row with a zero in the pivot column and on any column beyond the last to have an entry in the pivot row.

ATTRIBUTES — **Version:** 1.0.2. (28 March 2013) **Types:** Real (single, double). **Calls:** None. **Origin:** J.K. Reid, Rutherford Appleton Laboratory. **Original date:** January 2001.

2 HOW TO USE THE PACKAGE

2.1 Argument lists and calling sequences

There are two subroutines that can be called by the user:

(a) `MA65A/AD` factorizes a matrix **A**.

(b) `MA65B/BD` uses the factors generated by `MA65A/AD` to solve a system of equations $\mathbf{Ax}=\mathbf{b}$ or $\mathbf{A}^T\mathbf{x}=\mathbf{b}$.

A call to `MA65B/BD` must be preceded by a call to `MA65A/AD`. `MA65B/BD` may be used repeatedly to solve for different systems with the same matrix **A**.

2.1.1 To factorize a matrix

The single precision version

```
CALL MA65A(N,KL,KU,ROWEND,COLEND,A,THRESH,IPIV,INFO)
```

The double precision version

```
CALL MA65AD(N,KL,KU,ROWEND,COLEND,A,THRESH,IPIV,INFO)
```

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

`KL` is an `INTEGER` variable that must be set by the user to the lower semibandwidth kl. It is not altered by the subroutine. **Restriction:** `KL` ≥ 0.

`KU` is an `INTEGER` variable that must be set by the user to the upper semibandwidth ku. It is not altered by the subroutine. **Restriction:** `KU` ≥ 0.

`ROWEND` is an `INTEGER` array of size `N` that must be set by the user so that `ROWEND(I)` holds the column index of the last entry in row `I`, `I` = 1, 2, ..., `N`. On return, the array is revised to refer to the factorization. **Restrictions:** `I` ≤ `ROWEND(I)` ≤ `N`, `I` = 1, 2, ..., `N`.

`COLEND` is an `INTEGER` array of size `N` that must be set by the user so that `COLEND(J)` holds the row index of the last entry in column `J`, `J` = 1, 2, ..., `N`. On return, the array is revised to refer to the factorization. **Restrictions:** `J` ≤ `COLEND(J)` ≤ `N`, `J` = 1, 2, ..., `N`.

`A` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `N*(2*KL+KU+1)` that must be set by the user to

hold the matrix by rows. Each row is held contiguously with the diagonal entry of row I in A(I*KB-KU-KL), where KB = 2*KL+KU+1 is the bandwidth. Zeros beyond ROWEND(I) of row I or COLEND(J) of column J need not be set. On return, the array holds the factorization in the same format.

THRESH is a REAL (DOUBLE PRECISION in the D version) variable that must be set by the user to a threshold for the absolute value of an entry in the pivot column. Entries at or below this threshold are treated as zero. A suitable value is the round-off error in the smallest matrix entry. Zero may be specified, in which case only exact zeros are treated as zero. It is not altered by the subroutine. **Restriction:** THRESH ≥ 0.0.

IPIV is an INTEGER array of size N that need not be set by the user. On return, IPIV(I) holds the index of the row that was exchanged with row I before being used as pivot row, I = 1, 2, ..., N.

INFO is an INTEGER variable that need not be set by the user. On return, the value is zero after a successful factorization. Negative values of INFO indicate error returns with all input arguments unchanged. They are as follows:

–1 N < 1.

–2 KL < 0.

–3 KU < 0.

–4 ROWEND(I) < I for one or more values of I.

–5 ROWEND(I) > N for one or more values of I.

–6 COLEND(J) < J for one or more values of J.

–7 COLEND(J) > N for one or more values of J.

–8 THRESH < 0.0.

A positive value indicates that the matrix is singular and shows the pivot step at which singularity was detected. The arrays A, ROWEND and COLEND will have been altered and are not suitable for passing to MA65B/MA65BD.

2.1.2 To solve equations, given a successful factorization

The single precision version

```
CALL MA65B(N,KL,KU,ROWEND,COLEND,A,IPIV,TRANS,X)
```

The double precision version

```
CALL MA65BD(N,KL,KU,ROWEND,COLEND,A,IPIV,TRANS,X)
```

N is an INTEGER variable that holds order *n* of the matrix **A**. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

KL is an INTEGER variable that holds the lower semibandwidth *kl*. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

KU is an INTEGER variable that holds the upper semibandwidth *ku*. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

ROWEND is an INTEGER array of size N that holds the column indices of the last entries in the rows. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

COLEND is an INTEGER array of size N that holds the row indices of the last entries in the columns. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

A is a REAL (DOUBLE PRECISION in the D version) array of length N*(2*KL+KU+1) that holds the factorization. It must be unchanged since the call to MA65A/AD and is not altered by the subroutine.

IPIV is an INTEGER array of size N that holds the row interchanges. It must be unchanged since the call to MA65A/AD

and is not altered by the subroutine.

TRANS is a LOGICAL variable that must be set by the user to .TRUE. to solve $\mathbf{A}^T\mathbf{x}=\mathbf{b}$ or .FALSE. to solve $\mathbf{Ax}=\mathbf{b}$. It is not altered by the subroutine.

X is a REAL (DOUBLE PRECISION in the D version) array of size N that must be set by the user so that X(I) holds component I of the right-hand side of the equations being solved. On return, it will hold the corresponding component of the solution vector.

2.2 Ordering for small bandwidth

The efficiency of this code is very dependent on the ordering of the rows and columns. We have found that a good ordering is often obtained with the HSL code MC62. The example in Section 5.2 takes a sparse matrix in co-ordinate form, sorts it by rows, finds an ordering with MC62 and then calls MA65.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: None.

Input/output: None

Restrictions:

N ≥ 1.
KL ≥ 0.
KU ≥ 0.
I ≤ ROWEND(I) ≤ N, I = 1, 2, ..., N.
J ≤ COLEND(J) ≤ N, J = 1, 2, ..., N.
THRESH ≥ 0.0.

4 METHOD

Gaussian elimination with row interchanges is used. Each pivot is found by searching the pivot column for its largest entry and exchanging its row with the pivot row. For pivot J, the search can be confined to rows J to COLEND(J).

The diagonal entries are not required to be nonzero, but we do require the diagonal to be present explicitly (ROWEND(I) ≥ I, COLEND(J) ≥ J). This simplifies the logic and we expect most applications to be unaffected.

To simplify the row exchanges, MA65A/AD starts by increasing COLEND(J) to max(COLEND(1:J)), J = 1, 2, ..., N, that is, making COLEND monotonic. Corresponding explicit zeros are set in the array A. This ensures that all entries of each row from the first to the last are held explicitly. Apart from this, COLEND is not changed.

When rows are exchanged, the corresponding components of ROWEND must be exchanged. This may cause the non-pivot row to end ahead of its diagonal. In this case, we insert zeros explicitly as far as the diagonal and adjust ROWEND.

5 EXAMPLE OF USE

5.1 Solving a banded set of equations

We illustrate the use of the package on the solution of the single set of equations given by:

$$\begin{pmatrix} 2 & 3 & & & \\ 3 & 0 & 4 & 6 & \\ & 4 & 1 & 5 & \\ & & 5 & 0 & \\ & & & & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8 \\ 39 \\ 31 \\ 15 \\ 5 \end{pmatrix}$$

Program

```
C Simple example of use of MA65 package
      INTEGER LFACT, LIFACT, NEMAX, NMAX
      PARAMETER (KBMAX=10, NMAX=7)
      INTEGER N,KL,KU,ROWEND(NMAX),COLEND(NMAX),IPIV(NMAX),INFO
      DOUBLE PRECISION A(NMAX*KBMAX),THRESH,X(NMAX)
      INTEGER I,K,KB
      LOGICAL TRANS

C Read matrix and right-hand side
      READ (5,*) N,KL,KU
      KB = 2*KL+KU+1
      IF (N.GT.NMAX .OR. KB.GT.KBMAX) THEN
        WRITE(6,'(A)') ' Immediate return'
        IF (N.GT.NMAX) WRITE(6,'(A,I6,A)') ' N =',N,' is too large.'
        IF (KB.GT.KBMAX) WRITE(6,'(A,I6,A)') ' KB =',KB,' is too large.'
        STOP
      ENDIF
      K = KL + 1
      DO 10 I = 1, N
        READ (5,*) (A(K+J),J=-KL,KU),X(I)
        K = K + KB
   10 CONTINUE

C Set threshold for treating entries as zeros
      THRESH = 1.0D-16

C Find the final entries of the rows
      DO 40 I = 1, N
        L = MIN(N,I+KU)
        K = I*KB-KL-KU+L-I
        DO 20 J = L,I+1,-1
           IF(ABS(A(K)).GT.THRESH)GO TO 30
           K = K - 1
   20   CONTINUE
   30   ROWEND(I) = J
   40 CONTINUE

C Find the final entries of the columns
      DO 70 J = 1, N
        L = MIN(N,J+KL)
        K = (L-1)*KB+1
        DO 50 I = L,J+1,-1
           IF(ABS(A(K)).GT.THRESH)GO TO 60
           K = K - KB
   50   CONTINUE
   60   COLEND(J) = I
   70 CONTINUE
```

```
C Factorize the matrix
      CALL MA65AD(N,KL,KU,ROWEND,COLEND,A,THRESH,IPIV,INFO)

C Solve the equations
      TRANS = .FALSE.
      CALL MA65BD(N,KL,KU,ROWEND,COLEND,A,IPIV,TRANS,X)

C Print out solution vector.
      WRITE(6,'(/A/(1P,5D13.5))') ' The solution vector is:',
     *                           (X(I),I=1,N)
      END
```

Data

```
5 1 2
0 2 3 0 8
3 0 4 6 39
4 1 5 0 31
5 0 0 0 15
0 1 0 0 5
```

Output

```
 The solution vector is:
  1.00000D+00  2.00000D+00  3.00000D+00  4.00000D+00  5.00000D+00
```

5.2 Solving a sparse set of equations by reordering

We illustrate the use of the package on the solution of the single set of equations given by:

$$\begin{pmatrix} & & 3 & & 2 \\ & 0 & 4 & 6 & 3 \\ & 4 & 1 & 5 & \\ & & 5 & & \\ 1 & & & & \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8 \\ 39 \\ 31 \\ 15 \\ 5 \end{pmatrix}$$

which is a column permutation of the previous example.

Program

```
C Example using MC62 to reorder to band form
      INTEGER LA,LIW,MAXK,MAXN,MAXNE
      PARAMETER (LA=1000,MAXN=30,MAXK=6,MAXNE=200,LIW=9*MAXN+2*MAXNE)
      DOUBLE PRECISION A(LA),AA(MAXN*MAXK),B(MAXN),RINFO(30),
     *     X(MAXN),THRESH,WT(3)
      INTEGER COLEND(MAXN),FIRST(MAXN), I,II,IP(MAXN+1),IQ(MAXN+1),
     *        IRN(LA),IW(LIW),ICNTL(10),INFO(10),J,JCN(LA),JJ,
     *        K,KB,KK,KL,KU,N,NE,NEXT(MAXN),
     *        PERMC(MAXN),PERMR(MAXN),ROWEND(MAXN)
      LOGICAL LELIM(MAXN),TRANS

C     Read in input matrix and RHS
      READ (5,*) N,NE
      DO 10 K = 1, NE
         READ (5,*) A(K),IRN(K),JCN(K)
   10 CONTINUE
      READ (5,*) B(1:N)

C Sort matrix by rows
      ICNTL(1) = 0
      ICNTL(2) = 0
      ICNTL(3) = 0
      ICNTL(4) = 6
```

```
      ICNTL(5) = 6
      ICNTL(6) = 0
      CALL MC59AD( ICNTL, N, N, NE, JCN, LA, IRN, LA, A, MAXN+1,
     +      IQ, 2*MAXN, IW, INFO)

C Find good row permutation
      CALL MC62ID(ICNTL)
      ICNTL(2) = -1
      CALL MC62AD(N,N,NE,JCN,IQ,IRN,IP,LELIM,PERMR,
     *            WT,LIW,IW,X,ICNTL,INFO,RINFO)
      KL = RINFO(7) - 1

C Find the column ends in the new order
      DO 30 II = 1,N
        I = PERMR(II)
        DO 20 K = IQ(I),IQ(I+1)-1
           J = JCN(K)
           COLEND(J) = II
   20   CONTINUE
   30 CONTINUE

C Link columns by column ends in the new order
      DO 40 I = 1,N
        FIRST(I) = 0
   40 CONTINUE
      DO 50 J = 1,N
        I = COLEND(J)
        NEXT(J) = FIRST(I)
        FIRST(I) = J
   50 CONTINUE

C Order the columns by their ends in the new order
      K = 1
      DO 70 I = 1,N
        J = FIRST(I)
        DO 60 II = 1,N
          IF(J.EQ.0)EXIT
          PERMC(J) = K
          COLEND(K) = I
          K = K + 1
          J = NEXT(J)
   60   CONTINUE
   70 CONTINUE

C Find row ends in the new order
      KU = 0
      DO 90 II = 1,N
        I = PERMR(II)
        JJ = 0
        DO 80 K = IQ(I),IQ(I+1)-1
           JJ = MAX(PERMC(JCN(K)),JJ)
   80   CONTINUE
        ROWEND(II) = JJ
        KU = MAX(KU,JJ-II)
   90 CONTINUE

      WRITE(6,'(A,2i4)')' The semibandwidths are',KL,KU

      KB = 2*KL+KU+1
      IF(KB.GT.MAXK)THEN
        WRITE(*,*)'MAXK must be at least',KB
```

```
        STOP
      END IF

C Load the permuted matrix and RHS
      KK = KL + 1
      DO 120 II = 1,N
        DO 100K = KK-KL,KK+KU
           AA(K) = 0
 100    CONTINUE
        I = PERMR(II)
        X(II) = B(I)
        DO 110 K = IQ(I),IQ(I+1)-1
           AA(PERMC(JCN(K))-II+KK) = A(K)
 110    CONTINUE
        KK = KK + KB
 120  CONTINUE

C Solve
      THRESH = 0.0
      TRANS = .FALSE.
      CALL MA65AD(N,KL,KU,ROWEND,COLEND,AA,THRESH,IW,INFO(1))
      CALL MA65BD(N,KL,KU,ROWEND,COLEND,AA,IW,TRANS,X)

C Permute the solution and write it out
      DO 130 J = 1,N
        JJ = PERMC(J)
        B(J) = X(JJ)
 130  CONTINUE
      WRITE (6,'(/A,(5F10.3))') ' The solution is:',B(1:N)

      END
```

Data

```
5 10
1.0 5 1
3.0 1 2
4.0 3 2
4.0 2 3
1.0 3 3
5.0 4 3
6.0 2 4
5.0 3 4
2.0 1 5
3.0 2 5
8.0 39.0 31.0 15.0 5.0
```

Output

```
 The semibandwidths are   1   2

 The solution is:     5.000     2.000     3.000     4.000     1.000
```