1 SUMMARY

To **read a sparse matrix,** coded in a **Harwell-Boeing format** with possible right-hand sides. The subroutine reads assembled as well as unassembled matrices, and returns them in a column oriented compressed sparse format. If the matrix being read is complex `ME36A/AD` should be used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** `MC36A` and `MC36AD`. **Calls:** `FA04` and `MC34`. **Language:** Fortran 77. **Original date:** June 1995. **Origin:** I.S. Duff, Rutherford Appleton Laboratory. **Licence:** A third-party licence for this package is available without charge.

2 HOW TO USE THE PACKAGE

2.1 The argument list

There is a single subroutine for both assembled and unassembled matrices, with or without right-hand sides.

The single precision version

```
      CALL MC36A(NMAX,NZMAX,TITLE,ELT,NROW,NCOL,NNZERO,
     *           COLPTR,ROWIND,VALUES,MAXRHS,NRHS,NZRHS,RHSPTR,
     *           RHSIND,RHSVAL,GUESS,SOLN,IW,ICNTL,INFO)
```

The double precision version

```
      CALL MC36AD(NMAX,NZMAX,TITLE,ELT,NROW,NCOL,NNZERO,
     *            COLPTR,ROWIND,VALUES,MAXRHS,NRHS,NZRHS,RHSPTR,
     *            RHSIND,RHSVAL,GUESS,SOLN,IW,ICNTL,INFO)
```

`NMAX` is an `INTEGER` variable that must be set by the user and must be greater than the number of columns of the matrix to be read. This argument is not altered by the subroutine.

`NZMAX` is an `INTEGER` variable that must be set by the user and must be greater than the number of entries of the matrix to be read. This argument is not altered by the subroutine.

`TITLE` is a `CHARACTER` variable of length 80 that need not be set by the user. On exit, `TITLE` will contain the title of the matrix and its key.

`ELT` is a `LOGICAL` variable that need not be set by the user. On exit, it is `.TRUE.` if the matrix is in unassembled element form and is `.FALSE.` otherwise.

`NROW` is an `INTEGER` variable that will be set by the subroutine to the number of rows of the matrix.

`NCOL` is an `INTEGER` variable that will be set by the subroutine to the number of columns of the matrix. For unassembled element problems it will be set to number of elements.

`NNZERO` is an `INTEGER` variable that will be set by the subroutine to the number of entries of the matrix.

`COLPTR` is an `INTEGER` array of size `NMAX+1` that need not be set by the user. On exit, `COLPTR(I)`, `I=1`, ..., `NCOL`, will hold the position in `ROWIND` of the first entry in column `I`. `COLPTR(NCOL+1)-1` is the position of the last entry in column `NCOL`. For unassembled element problems (`ELT=.TRUE.`), `COLPTR` will point to the start of each element in array `ROWIND`.

`ROWIND` is an `INTEGER` array of size `NZMAX` that need not be set by the user. On exit, the first `NNZERO` positions will be set to the row indices of the entries, ordered by columns. For unassembled element problems, it will be set to the indices of the variables of each element.

VALUES is a REAL (DOUBLE PRECISION in the D version) array of size NZMAX that need not be set by the user. On exit, the first NNZERO positions will contain the numerical values of the entries. For unassembled element problems, it will contain contiguously the numerical values of each element held by columns. If the elements are symmetric, then VALUES will hold the lower triangle by columns. If ICNTL(4) $\neq$ 0 or 1, VALUES is not accessed and so could be a dummy array of length 1.

MAXRHS is an INTEGER variable that must be set by the user to the maximum number of right-hand sides. A zero or negative value means that no right-hand sides will be read.

NRHS is an INTEGER variable that need not be set by the user. On exit, it will hold the actual number of right-hand sides read. No right-hand sides will be read if ICNTL(4) $\neq$ 0 or 1, or if MAXRHS = 0.

NZRHS is an INTEGER variable that must be set by the user to the length of arrays RHSIND and RHSVAL. Normally this will be much less than NZMAX but, in the event that this is set too low, a suitable value will be output in INFO(7).

RHSPTR is an INTEGER array that need not be set by the user. It is only accessed if right-hand sides are read and the right-hand sides are held in sparse format. If the right-hand sides are held in sparse format, RHSPTR(I) will, on exit, be set to the position of the first entry in arrays RHSIND and RHSVAL of right-hand side I, I=1, ..., NRHS, and RHSPTR(NRHS+1) will be set to the position immediately after the last entry in arrays RHSIND and RHSVAL.

RHSIND is an INTEGER array of length NZRHS that need not be set by the user and will be set to hold the indices of the right-hand sides. It is only accessed if right-hand sides are read (NRHS > 1). If the right-hand sides are held as full vectors, RHSIND(1) is set to –1, and no other entry of RHSIND is accessed. If the right-hand sides are in elemental form, RHSIND(1) is not accessed.

RHSVAL is a REAL (DOUBLE PRECISION in the D version) array of length NZRHS that need not be set by the user and will be set to the right-hand side values. It is only accessed if right-hand sides are read (NRHS > 1). If the matrix is assembled (ELT=.FALSE.), then each right-hand side is held contiguously. If the right-hand sides are held in sparse format, RHSVAL(K) is the value of component RHSIND(K), K=1, ..., RHSPTR(NRHS+1)-1. If the matrix is an unassembled element matrix (ELT=.TRUE.), the entries of the right-hand sides for element 1 immediately precede these for element 2 and so on. In each case, the components for each right-hand side are held contiguously.

GUESS is a REAL (DOUBLE PRECISION in the D version) array that need not be set by the user. If INFO(2)=1, the first NRHS*NROW entries will be set so that the I th component of guess J is in position GUESS((J-1)*NROW+I).

SOLN is a REAL (DOUBLE PRECISION in the D version) array that need not be set by the user. If INFO(3)=1, the first NRHS*NROW entries will be set so that the I th component of solution J is in position SOLN((J-1)*NROW+I).

IW is an INTEGER array of size NMAX that need not be set by the user and is used as a work array.

ICNTL is an INTEGER array of length 10 that contains control parameters and must be set by the user. Details of the control parameters are given in Section 2.2. This array is not altered by the subroutine.

INFO is an INTEGER array of length 10 that need not be set by the user. On a successful return, INFO(1) is set to zero. For nonzero values, see Section 2.3. For the meaning of the value of other components of INFO, see Section 2.2.

2.2 Arrays for control and information

The components of the array ICNTL control the action of MC36A/AD and the first four components must be set by the user before calling MC36A/AD. The remaining components are for possible future use and are currently not accessed. They are not altered by MC36A/AD. The components of the array INFO provide information on the action of MC36A/AD. They need not be set by the user.

ICNTL(1) must be set by the user to the unit number from which the matrix is to be read.

ICNTL(2) must be set by the user to the unit number of the device to which messages are sent. These messages include data on the matrix as well as error messages. Messages are suppressed if `ICNTL(2)` < `0`.

`ICNTL(3)` is used to indicate whether both triangles of a symmetric matrix will be held. If `ICNTL(3)` has the value `1` and the matrix is symmetric, only the lower triangular part of the matrix is stored. Otherwise, both lower and upper triangles are held. `ICNTL(3)` is ignored if the matrix is unsymmetric.

`ICNTL(4)` must be set to `0` or `1` if the user wants numerical values. If `ICNTL(4)=1`, in cases where these are not provided in the Harwell-Boeing input, the values are generated by the random number routine `FA04`. If `ICNTL(4)=0`, real values will only be provided if they are present in the input data set. A value for `ICNTL(4)` other than `0` or `1` will result in reals not being read or generated.

`ICNTL(5)` to `ICNTL(10)` are not currently used and are not accessed by the routine.

`INFO(1)` is set to `0` on successful exit and to a negative value in the event of an error (see Section 2.3).

`INFO(2)` is set to `1` if the matrix is symmetric, `2` if the matrix is skew symmetric, or to `0` otherwise.

`INFO(3)` is set to `1` if there are starting guesses or to `0` otherwise.

`INFO(4)` is set to `1` if there are exact solutions given or to `0` otherwise.

`INFO(5)` is set to `1` if reals have been read or and to `2` if they have been generated using `FA04`. If reals are not returned, it is set to `0`.

`INFO(6)` is set to the number of right-hand sides if `MAXRHS` was not large enough to read them (see error `INFO(1)=-5`) or if `MAXRHS=0`.

`INFO(7)` is set to the length of the arrays required to hold the right-hand sides (see error `INFO(1)=-6`).

`INFO(8)` to `INFO(10)` are set to zero.

2.3 Error diagnostics

A successful return from `MC36A/AD` is indicated by a value of `INFO(1)` equal to zero. A negative value of `INFO(1)` is associated with an error message that will be output on unit `ICNTL(2)`.

–1 The matrix has more columns than `NMAX`. The number is given in `NCOL`.

–2 The matrix has more entries than `NZMAX`. The number is given in `NNZERO`.

–3 The matrix is complex. Subroutine `ME36A/AD` should be used.

–4 There is an error in the matrix data file. The matrix is not in Harwell-Boeing format.

–5 There are more right-hand sides than `MAXRHS` (with `MAXRHS` > `0`). The number is given in `INFO(6)`. In this case, the matrix will have been read successfully.

–6 The right-hand side arrays are of insufficient length. `NZRHS` must be increased to the value given in `INFO(7)`. The number of right-hand sides that the subroutine was trying to read is given in `INFO(6)`. In this case, the matrix will have been read successfully.

–7 Return because at end of input file or input file null.

3 GENERAL INFORMATION

Workspace: `IW` of length `NMAX`.

Use of common: None.

Other routines called directly: `MC36A/AD` calls internal routines `MC36B/BD` and `MC36C/CD` that need not be called by the user. The subroutines `FA04A/AD` and `MC34A/AD` from the Harwell Subroutine Library are also called.

Input/output: Error messages and information on files are written to stream `ICNTL(2)`.

3.1 Storage of sparse matrices.

We describe here the Harwell-Boeing storage format as described in the Users' guide of the Harwell-Boeing collection.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1989). Sparse matrix test problems. *ACM Trans. Math. Softw.* **15**, 1-14.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1992). Users' Guide for the Harwell-Boeing Sparse Matrix Collection. Report RAL 92-086, Rutherford Appleton Laboratory, Chilton, Didcot,Oxfordshire OX11 0QX.

3.1.1 Standard sparse matrix format

The standard sparse matrix format is column-oriented. That is, the matrix is represented by a sequence of columns. Each column is held as a sparse vector, represented by a list of the row indices of the entries in an integer array and a list of the corresponding values in a separate real array. A single integer array and a single real array are used to store the row indices and the values, respectively, for all of the columns. (Throughout, we use the term "real" in a generic sense so that it should be read as a Fortran real, double precision, complex, or double precision complex as appropriate). Data for each column are stored in consecutive locations, the columns are stored in order, and there is no space between the columns. A separate integer array holds the location of the first entry of each column and the first free location. For symmetric and Hermitian matrices, we store only the entries of the lower triangle (including the diagonal). For skew symmetric matrices, we hold only the strict lower triangle.

We illustrate the storage scheme with the following example. The 5×5 matrix

$$\begin{pmatrix} 1. & -3. & 0 & -1. & 0 \\ 0 & 0 & -2. & 0 & 3. \\ 2. & 0 & 0 & 0 & 0 \\ 0 & 4. & 0 & -4. & 0 \\ 5. & 0 & -5. & 0 & 6. \end{pmatrix}$$

would be stored in the arrays `COLPTR` (location of first entry), `ROWIND` (row indices), and `VALUES` (numerical values) according to the following prescription:

Subscripts	1	2	3	4	5	6	7	8	9	10	11
`COLPTR`	1	4	6	8	10	12					
`ROWIND`	1	3	5	1	4	2	5	1	4	2	5
`VALUES`	1.	2.	5.	–3.	4.	–2.	–5.	–1.	–4.	3.	6.

We can generate column 5, say, by observing that its first entry is in position `COLPTR(5)` = 10 of arrays `ROWIND` and `VALUES.` This entry is in row `ROWIND(10)` = 2 and has value `VALUES(10) = 3.` Other entries in column 5 are found by scanning `ROWIND` and `VALUES` to position `COLPTR(6)-1,` that is position 11. Thus, the only other entry in column 5 is in row `ROWIND(11) = 5` with value `VALUES(11)` = `6`.

3.1.2 Finite-element matrices in unassembled format

Matrices arising in finite-element applications are usually assembled from numerous small elemental matrices. Our collection includes a few sparse matrices in original unassembled form. The storage of the unassembled matrices is analogous to the explicit form above, which stores each matrix as a list of matrix columns. The elemental representation stores the matrix as a list of elemental matrices. Each elemental matrix is represented by a list of the row/column indices (variables) associated with the element and by a small dense matrix giving the numerical values by columns (in the symmetric case only the lower triangular part). The lists of indices are held contiguously, just as for the lists of row indices in the standard format. The dense matrices are held contiguously in a separate array, with each matrix held by columns. Although there is not a 1-1 correspondence between the arrays of integer and numerical values, our representation does not hold the pointers to the beginning of the real values for each element. These pointers can be created from the index start pointers (`ELTPTR`) after noting that an element with v variables has v^2 real values ($v \times (v+1) / 2$ in the symmetric case).

We illustrate the elemental storage scheme with a small example. Consider a 5×5 symmetric matrix

$$\begin{pmatrix} 5. & 0. & 0. & 1. & 2. \\ 0. & 4. & 3. & 0. & 6. \\ 0. & 3. & 7. & 8. & 1. \\ 1. & 0. & 8. & 9. & 0. \\ 2. & 6. & 1. & 0. & 10. \end{pmatrix},$$

generated from four elemental matrices,

$$\begin{array}{c} 1 \\ 4 \end{array}\begin{pmatrix} 2. & 1. \\ 1. & 7. \end{pmatrix} \qquad \begin{array}{c} 1 \\ 5 \end{array}\begin{pmatrix} 3. & 2. \\ 2. & 8. \end{pmatrix} \qquad \begin{array}{c} 2 \\ 3 \\ 5 \end{array}\begin{pmatrix} 4. & 3. & 6. \\ 3. & 5. & 1. \\ 6. & 1. & 2. \end{pmatrix} \qquad \begin{array}{c} 3 \\ 4 \end{array}\begin{pmatrix} 2. & 8. \\ 8. & 2. \end{pmatrix},$$

where the variable indices are indicated by the integers before each matrix (columns are identified symmetrically to rows). This matrix would be stored in the arrays `ELTPTR` (location of first entry), `VARIND` (variable indices), and `VALUES` (numerical values) according to the following prescription:

Subscripts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`ELTPTR`	1	3	5	8	10										
`VARIND`	1	4	1	5	2	3	5	3	4						
`VALUES`	2.	1.	7.	3.	2.	8.	4.	3.	6.	5.	1.	2.	2.	8.	2.

3.1.3 Right-hand sides

Where the matrices originate in the solution of linear equations and the right-hand sides are available, the right-hand-side vectors are stored with the matrices. Usually the right-hand-side vectors are dense, in which case they are stored contiguously (as in ordinary Fortran array storage). Multiple right-hand sides are stored as consecutive vectors, so that the right-hand sides are accessible as the columns of a Fortran array.

An alternative form is available in which right-hand sides are represented in the same format as the matrix. For unassembled matrices the associated right-hand sides can be represented by elemental contributions. Right-hand sides in elemental form are stored as a sequence of small dense matrices, each small matrix having as many columns as the number of right-hand sides, and with as many rows as the corresponding element in the matrix representation. Within each elemental right-hand side, the rows correspond to the entries in the variable index vector for that element.

The format for assembled sparse matrices is used to store sparse right-hand sides. Applications with sparse right-hand sides are less common, but the sparsity can be used to advantage in direct solution techniques. We only allow sparse right-hand sides for assembled matrices, in which case we store the right-hand sides exactly as a standard sparse matrix, with the same number of rows as the coefficient matrix and the same number of columns as right-hand sides.

We allow the specification of a starting guess for the solution of the problem and a vector that purports to be the exact solution. These can only be supplied as full arrays and only when right-hand sides are present. Either or both of these arrays can be present. The starting vectors precede the solution vectors if both are given and the number of such vectors must be equal to the number of right-hand sides.

4 METHOD

The code reads sequentially the data from the unit `ICNTL(1)`. It checks the sizes, and, if values are wanted but not provided, it fills the pattern with random numbers using routine `FA04A/AD` of the Harwell Subroutine Library. The subroutine can read a symmetric matrix stored as a lower triangle and store it in the same data structure as an unsymmetric matrix. This is done by a call to `MC34A/AD`.

5 EXAMPLES OF USE

We illustrate the use of the subroutine by a small program that reads a matrix from a file and prints it out column by column.

```
      PROGRAM READ

C   Maximum size for arrays
      INTEGER NMAX,NZMAX,MAXRHS,NZRHS
      PARAMETER (NMAX=100, NZMAX=4000, MAXRHS=0, NZRHS=1)

C   MC36 parameters
      INTEGER NROW, NCOL, NNZERO, ICNTL(10)
      CHARACTER TITLE*80
      INTEGER COLPTR(NMAX+1), ROWIND(NZMAX)
      DOUBLE PRECISION VALUES(NZMAX), RHSVAL(NZRHS),
     *       GUESS(NMAX*MAXRHS+1), SOLN(NMAX*MAXRHS+1)
      INTEGER  RHSPTR(MAXRHS+1), RHSIND(NZRHS)
      LOGICAL ELT
      INTEGER NRHS,IW(NZMAX),INFO(10)

C   Local variables
      INTEGER I,J,J1,J2
```

```
C    Read the matrix
      ICNTL(1) = 5
      ICNTL(2) = 6
      ICNTL(3) = 2
      ICNTL(4) = 1
      CALL MC36AD(NMAX,NZMAX,TITLE,ELT,NROW,NCOL,NNZERO,
     1            COLPTR,ROWIND,VALUES,MAXRHS,NRHS,NZRHS,
     2            RHSPTR,RHSIND,RHSVAL,GUESS,SOLN,IW,ICNTL,INFO)

C  The matrix is now output

       DO 20 J=1,NCOL
         WRITE(6,'(A,I3)') 'Row indices and values in column',J
         J1 = COLPTR(J)
         J2 = COLPTR(J+1)-1
         WRITE(6,'(8(I2,F6.2,2X))') (ROWIND(I),VALUES(I),I=J1,J2)
   20    CONTINUE

       END
```

If we run on Matrix JGL009 from the counterx.data set of the Harwell-Boeing Collection that is held as:

```
1U JOHN G. LEWIS P4 COUNTEREXAMPLE WHICH REQUIRES FILL IN SPIKES          JGL009
             3             1             2             0             0
PUA                       9             9            50             0
(24I3)          (36I2)          (4E20.12)
  1  9 13 21 27 33 39 44 46 51
 1 2 4 5 6 7 8 9 2 3 8 9 2 3 4 5 6 7 8 9 4 5 6 7 8 9 4 5 6 7 8 9 4 5 6 7
 8 9 1 2 3 8 9 8 9 1 2 3 8 9
```

It produces :

```
1U JOHN G. LEWIS P4 COUNTEREXAMPLE WHICH REQUIRES FILL IN SPIKES          JGL009
 Order of matrix is            9
 Number of entries is         50
Row indices and values in column  1
 1 -1.00   2 -0.74   4  0.51   5 -0.08   6  0.07   7 -0.56   8 -0.91   9  0.36
Row indices and values in column  2
 2  0.36   3  0.87   8 -0.23   9  0.04
Row indices and values in column  3
 2  0.66   3 -0.93   4 -0.89   5  0.06   6  0.34   7 -0.98   8 -0.23   9 -0.87
Row indices and values in column  4
 4 -0.17   5  0.37   6  0.18   7  0.86   8  0.69   9  0.05
Row indices and values in column  5
 4 -0.82   5  0.31   6 -0.17   7  0.40   8  0.82   9  0.52
Row indices and values in column  6
 4 -0.48   5 -0.91   6  0.47   7 -0.34   8  0.27   9  0.51
Row indices and values in column  7
 1  0.98   2 -0.27   3 -0.51   8  0.97   9  0.45
Row indices and values in column  8
 8  0.51   9  0.30
Row indices and values in column  9
 1 -0.85   2  0.26   3  0.77   8 -0.45   9 -0.13
```