1 SUMMARY

This subroutine **calculates scaling factors for a sparse matrix A** = $\{a_{ij}\}_{m \times n}$**.** They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the entries is minimized. The natural logarithms of the scaling factors r_i, $i = 1, 2,..., m$ for the rows and c_j, $j = 1, 2,..., n$ for the columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(r_i + c_j).$$

The method is described by Curtis and Reid, J. Inst. Maths. Applics. (1972), **10**, pp. 118-124.

We recommend that this package (or its companion `MC30` for symmetric scaling) be considered in conjunction with any of the HSL packages for the direct solution of linear sets of equations unless the matrix is symmetric and positive definite. This is because the pivoting strategies are based on the aim of finding a solution that is exact for a perturbed system

$$(\mathbf{A} + \delta\mathbf{A})\mathbf{x} = \mathbf{b} + \delta\mathbf{b}$$

where all the entries of $\delta\mathbf{A}$ are small compared with $\max|a_{ij}|$. Pivoting will not be successful if changes that are small compared with $\max|a_{ij}|$ cannot be tolerated, for example, if mixed units are in use so that the entries of some columns are much smaller than those of others. Scaling should really be applied to the errors that can be tolerated in the elements of **A**, but these are probably unknown. If all the entries are known to the same relative accuracy, scaling **A** itself has the same effect and `MC29` is likely to be very helpful. In other cases, it may give undue weight to small entries that are not known accurately.

ATTRIBUTES — **Version:** 1.0.0. (12 July 2004) **Types:** Real (single, double). **Remark:** Supersedes `MC19`. **Original date:** March 1993. **Origin:** J. K. Reid, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Argument list

The single precision version

        CALL MC29A(M,N,NE,A,IRN,ICN,R,C,W,LP,IFAIL)

The double precision version

        CALL MC29AD(M,N,NE,A,IRN,ICN,R,C,W,LP,IFAIL)

M   is an `INTEGER` variable that must be set by the user to the number *m* of rows. `M` is not altered by `MC29A/AD`. **Restriction:** `M` ≥ 1.

N   is an `INTEGER` variable that must be set by the user to the number *n* of columns. `N` is not altered by `MC29A/AD`. **Restriction:** `N` ≥ 1.

NE   is an `INTEGER` variable that must be set by the user to the number of entries in **A**. `NE` is not altered by `MC29A/AD`. **Restriction:** `NE` ≥ 1.

A   is a `REAL` array (`DOUBLE PRECISION` in the D version) of length `NE` that must be set by the user to hold the values of the entries of the matrix **A**. The entries may be in any order and the subroutine does not change their values.

IRN,ICN are two `INTEGER` arrays of length `NE` that the user must set to the row and column indices of the entries. If the matrix entry a_{ij} is held in `A(K)` then `IRN(K)` must contain *i* and `ICN(K)` must contain *j*. If `IRN(K)` or

`ICN(K)` is out of range, the entry is ignored. These arrays are not altered by the subroutine.

R   is a `REAL` array (`DOUBLE PRECISION` in the D version) of length m that need not be set by the user. On return `R(i)` contains r_i, $i = 1, 2, ..., m$.

C   is a `REAL` array (`DOUBLE PRECISION` in the D version) of length n that need not be set by the user. On return `C(j)` contains c_j, $j = 1, 2, ..., n$.

W   is a `REAL` array (`DOUBLE PRECISION` in the D version) of length $2m + 3n$ used for workspace.

LP   is an `INTEGER` variable that must be set by the user to specify the unit number to be used for the error messages, or zero to suppress the printing of error messages. `LP` is not altered by `MC29A/AD`.

`IFAIL` is an `INTEGER` variable that need not be set by the user. It is set by the subroutine to indicate success or failure. On exit from the subroutine, `IFAIL` will take one of the following values.

     0   successful entry,
   –1   `M` < 1 or `N` < 1.
   –2   `NE` < 1.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: None.

Input/output: in the event of errors, diagnostic messages are output on unit `LP`.

Restrictions: `M` ≥ 1, `N` ≥ 1, `NE` ≥ 1.

4 METHOD

The variables r_i and s_j are chosen to minimize the function

$$\Phi = \sum_{i,j} (f_{ij} + r_i + c_j)^2$$

where

$$f_{ij} = \log|a_{ij}|$$

and the summation is over pairs i, j for which $a_{ij} \neq 0$. This is done to sufficient accuracy in only a few matrix-by-vector multiplications. For further information, see Curtis and Reid, On the Automatic Scaling of Matrices for Gaussian Elimination, J. Inst. Maths. Applics. (1972), **10**, pp. 118-124.

Use of this method gives far better results on sparse matrices than scaling to equilibrate row norms.

5 EXAMPLE OF USE

The following program reads a sparse matrix, scales it and prints the result.

```
      DOUBLE PRECISION A(1000),C(100),R(100),W(500)
      INTEGER IRN(1000),ICN(1000),I,J,IFAIL,K,LP,M,N,NE
      PARAMETER (LP=6)

C  Read size and number of entries
      READ (5,*) M,N,NE
C  Check that these are within bounds
      IF (M.LE.0 .OR. M.GT.100) GO TO 40
      IF (N.LE.0 .OR. N.GT.100) GO TO 40
      IF (NE.LE.0 .OR. NE.GT.1000) GO TO 40
```

```
C  Read matrix entries and call MC29
      READ (5,*) (IRN(I),ICN(I),A(I),I=1,NE)
      CALL MC29AD(M,N,NE,A,IRN,ICN,R,C,W,LP,IFAIL)
C
C  Calculate scaling multipliers
      DO 10 I = 1,M
         R(I) = EXP(R(I))
   10 CONTINUE
      DO 20 J = 1,N
         C(J) = EXP(C(J))
   20 CONTINUE
C
C  Scale the matrix and print it
      DO 30 K = 1,NE
         A(K) = A(K)*R(IRN(K))*C(ICN(K))
   30 CONTINUE
      WRITE (6,'(2I5,F10.4)') (IRN(I),ICN(I),A(I),I=1,NE)
      STOP
C
C  Deal with error condition
   40 WRITE (6,'(A)') ' M, N or NE out of permitted range'
      END
```

To scale the following matrix

$$\begin{pmatrix} 100. & 0. & 0. \\ 0. & 6. & 0. \\ 900. & 0. & 110000. \\ 0. & 14000. & 16000. \end{pmatrix}$$

we could have as input

```
    4        3         6
    4        3     16000.
    1        1       100.
    4        2     14000.
    2        2         6.
    3        1       900.
    3        3    110000.
```

and we would get the following output

```
    4     3     0.8790
    1     1     1.8123
    4     2     1.2811
    2     2     0.7806
    3     1     0.5518
    3     3     1.1377
```

which corresponds to the scaled matrix

$$\begin{pmatrix} 1.8123 & 0. & 0. \\ 0. & 0.7806 & 0. \\ 0.5518 & 0. & 1.1377 \\ 0. & 1.2811 & 0.8790 \end{pmatrix}.$$