1 SUMMARY

This subroutine performs iterative refinement and provides information on the error in the resulting solution of a set of n sparse linear equations $\mathbf{A}\,\mathbf{x}=\mathbf{b}$, either as an upper bound on the actual error (**forward error**) or as a bound on the perturbation to the matrix elements which would make the solution obtained the exact solution of a perturbed problem with the same sparsity pattern as the original matrix (**backward error**). All estimates are provided in the ∞–norm, that is $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

There is an option for avoiding iterative refinement, in which case bounds on the errors associated with the given solution are provided.

ATTRIBUTES — **Version:** 1.2.0. (4 September 2007) **Types:** Real (single, double). **Calls:** `MC71`, `FD15`, `I_AMAX`. **Original date:** July 2001. **Remark:** `MA60` is a threadsafe version of `MA40`. **Origin:** M. Arioli, I.S. Duff, Harwell.

2 HOW TO USE THE PACKAGE

'Reverse communication' is used to provide the subroutine with the values $\mathbf{A}^{-1}\mathbf{y}$ or $\mathbf{A}^{-T}\mathbf{y}$ for a given vector $\mathbf{y}$, which means that control is returned to the user to compute these quantities. When the subroutine requires the values of these products, it sets the appropriate values in the array `Y` and returns to the user's program with a flag, called `KASE`, set to 1 if the product of `Y` by $\mathbf{A}^{-T}$ is required or set to 2 if the product of `Y` by $\mathbf{A}^{-1}$ is required. The user must set `Y` to the value of the matrix-vector product required by the subroutine and must not alter the value of any other arguments of the subroutine. Initially, the user must set the value of `KASE` to 0; the subroutine sets `KASE` to 0 at the end of the computation. An example of use is shown in Section 5.

2.1 Notation and theory

The subroutine uses the following iterative refinement scheme to improve the quality of the solution:

(a) compute the residual $\mathbf{r}=\mathbf{A}\,\tilde{\mathbf{x}}-\mathbf{b}$ without double-length accumulation of inner products;

(b) solve $\mathbf{A}\,\mathbf{d}=\mathbf{r}$ for $\mathbf{d}$; and

(c) update $\tilde{\mathbf{x}}=\tilde{\mathbf{x}}-\mathbf{d}$.

These three steps are repeated until a satisfactory error is obtained or the number of cycles of iterative refinement is greater than the maximum allowed.

The backward error can be measured componentwise so that the solution is an exact solution of the perturbed problem

$$(\mathbf{A}+\delta\mathbf{A})\,\mathbf{x}=\mathbf{b}+\delta\mathbf{b} \qquad \text{where} \qquad |\delta\mathbf{A}|\le\omega|\mathbf{A}|,\ |\delta\mathbf{b}|\le\omega|\mathbf{b}|$$

and the modulus sign $|\cdot|$ on an array produces a new array with entries equal to the absolute values of the entries in the old one. This measure maintains sparsity in $\mathbf{A}$, but usually it is impossible to satisfy the bounds on $|\delta\mathbf{b}|$ for all the components where b_i is at or near zero. We therefore use work by Arioli, Demmel and Duff (1988) to generalize these bounds by dividing the equations into two categories and obtaining bounds of the form

$$|\delta\mathbf{A}|\le\begin{pmatrix}\omega_1|\mathbf{A}^{(1)}|\\ \omega_2|\mathbf{A}^{(2)}|\end{pmatrix} \qquad \text{and} \qquad |\delta\mathbf{b}|\le\begin{pmatrix}\omega_1|\mathbf{b}^{(1)}|\\ \omega_2|\mathbf{A}^{(2)}|(|\tilde{\mathbf{x}}|+\mathbf{e}\|\tilde{\mathbf{x}}\|_\infty)\end{pmatrix},$$

where $\mathbf{e}$ is the column vector of all ones, the rows of $\mathbf{A}$ corresponding to the two categories are denoted by $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ respectively, and the components of $\mathbf{b}$ are denoted similarly.

Corresponding to each category there is a condition number, denoted by κ_{ω_1} and κ_{ω_2}, so that the forward error

$\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty}$ is bounded by $\omega_1 \kappa_{\omega_1}+\omega_2 \kappa_{\omega_2}$.

We provide ω_1 and ω_2 and the bound on the forward error to the user.

Further details on the use of the two categories can be found in Sections 2.4 and 4 and in the references therein.

2.2 The argument lists

2.2.1 The initialization entry

The MA60I/ID entry must be called before any calls are made to the MA60A/AD entry. It initializes the user options in ICNTL and the private workspace.

The single precision version

```
CALL MA60I(ICNTL,KEEP,RKEEP)
```

The double precision version

```
CALL MA60ID(ICNTL,KEEP,RKEEP)
```

ICNTL is the INTEGER array of length 5 that will be passed later to the MA60A/AD entry. On return it will contain default values, see the description of ICNTL in Section 2.2.2.

KEEP is the INTEGER array of length 10 that will be passed later to the MA60A/AD entry to be used as private integer workspace and must not be altered by the user.

RKEEP is the REAL (DOUBLE PRECISION in the D version) array of length 10 that will be passed later to the MA60A/AD entry to be used as private real workspace and must not be altered by the user.

2.2.2 The main emtry

The single precision version:

```
   CALL MA60A(N,NZ,A,IRN,JCN,RHS,X,Y,D,W,IW,KASE,OMEGA,ERX,JOB,
  +           COND,NOITER,ICNTL,KEEP,RKEEP)
```

The double precision version:

```
   CALL MA60AD(N,NZ,A,IRN,JCN,RHS,X,Y,D,W,IW,KASE,OMEGA,ERX,JOB,
  +            COND,NOITER,ICNTL,KEEP,RKEEP)
```

N is an INTEGER variable which must be set by the user to the order of the matrix. This argument is not altered by the subroutine. **Restriction:** N > 0.

NZ is an INTEGER variable which must be set by the user to the number of entries of the matrix. This argument is not altered by the subroutine. **Restriction:** NZ > 0.

A is a REAL (DOUBLE PRECISION in the D version) array of length NZ that must be set by the user to the nonzero elements of the original matrix. These elements can be in any order. This argument is not altered by the subroutine.

IRN is an INTEGER array of length NZ that must be set by the user to the row indices of the nonzero elements as ordered in A. This argument is not altered by the subroutine.

JCN is an INTEGER array of length NZ. JCN must hold the column indices of the matrix nonzero elements as ordered in A. This argument is not altered by the subroutine.

RHS is a REAL (DOUBLE PRECISION in the D version) array of length N that must be set by the user to the vector **b**. This argument is not altered by the subroutine.

X is a REAL (DOUBLE PRECISION in the D version) array of length N which must be set by the user to contain the

value of the initial guess of the solution. Unless the user has a priori information on the solution, we suggest an initial guess equal to the solution obtained by one application of the solver. However, a guess of all zeros is possible. On exit, X contains the solution with the best backward error obtained.

Y is a REAL (DOUBLE PRECISION in the D version) array of length N that need not be set by the user. For the intermediate returns the subroutine sets the value of Y and the user must overwrite it by the product of Y by the matrix $\mathbf{A}^{-T}$ (KASE = 1) or by the product of Y by the matrix $\mathbf{A}^{-1}$ (KASE = 2).

D is a REAL (DOUBLE PRECISION in the D version) array of length N. Normally the user must set entries of D to the value 1, but see Section 2.4 for further details.

W is a REAL (DOUBLE PRECISION in the D version) array of length 3N that is used as workspace.

IW is an INTEGER array of length 2N that is used as workspace.

KASE is an INTEGER variable that must be set by the user to 0 on the initial call. The subroutine sets KASE to 1 or 2 in the intermediate returns and to 0 for the final return. Negative values of KASE indicate an error condition (see Section 2.3).
When KASE = 1 the user must supply the product of Y by the matrix $\mathbf{A}^{-T}$.
When KASE = 2 the user must supply the product of Y by the transpose of the matrix $\mathbf{A}^{-1}$.

OMEGA is a REAL (DOUBLE PRECISION in the D version) array of length 2 that need not be set by the user. It will be set by the subroutine to the values of ω_1 and ω_2.

ERX is a REAL (DOUBLE PRECISION in the D version) variable that need not be set by the user. It will be set by the subroutine to the value of the estimate of the error in the solution if JOB is greater than zero.

JOB is an INTEGER array of length 2 which must be set by the user. If JOB(1) is negative or zero only the backward error estimate is made. If JOB(2) is set to one, it is assumed the matrix is symmetric.

COND is a REAL (DOUBLE PRECISION in the D version) array of length 2 which is set by MA60A/A to the values of the condition numbers κ_{ω_1} and κ_{ω_2}, see Section 2.1.

NOITER is an INTEGER variable which is set by MA60A/AD to the number of iterative refinement iterations actually used.

ICNTL is an INTEGER array of length 5 in which the user can set optional control information.

ICNTL(1) (default value is 6) is used by the subroutine as the unit number for its warning messages. The user can either reset it to a different unit number or suppress the output by setting it to zero.

ICNTL(2) (default value is 16) is the maximum number of iterations to be performed by the iterative refinement. If more than ICNTL(2) iterations are required KASE is set to -3.

KEEP is an INTEGER array of length 10 used for private integer workspace. It must be intialized by a call to MA60I/ID and thereafter must not be altered by the user.

RKEEP is an REAL (DOUBLE PRECISION in the D version) array of length 10 used for private real workspace. It must be intialized by a call to MA60I/ID and thereafter must not be altered by the user.

2.3 Errors and diagnostic messages

KASE = −1 indicates that N ≤ 0

KASE = −2 indicates that NZ ≤ 0.

KASE = −3 indicates that more than MAXIT iterations are required.

2.4 Further facilities

In this section we describe some other features connected with a more sophisticated use of the subroutine.

If the user is using a column scaling at step *(b)* of the iterative refinement described in Section 2.1, the entries of the

array D must be set to the values of this column scaling. Moreover, if the user is not interested in the error relative to particular components of **x**, the corresponding entries of the array D should be set to zero. The final estimate of the error will be relative to the components of **x** for which D is nonzero.

The first N entries of the workspace array IW give information about the equations in category 1 and category 2. They are equal to 1 if the corresponding equation is in category 1, or equal to 2 if the corresponding equation is in category 2.

Finally, the user can have an estimate of the error for the initial guess of the solution by setting MAXIT to zero and JOB(1) to a value greater than zero.

3 GENERAL INFORMATION

Workspace: Provided by the user, see arguments IW, W, KEEP and RKEEP.

Use of common: None.

Other routines called directly: MA60A/AD calls MA60B/BD, MA60C/CD, MA60F/FD, MC71A/AD and ISAMAX/IDAMAX. The user does not need to call these subroutines directly.

Input/output: Error and warning messages only on unit ICNTL(1).

Restrictions: N > 0 and NZ > 0.

4 METHOD

When solving sparse linear systems, it is desirable to produce the solution of a nearby sparse problem with the same sparsity structure. This kind of backward stability helps guarantee, for example, that one has solved a problem with the same physical connectivity as the original problem. Theorems of Oettli and Prager (1964) and Skeel (1980) show that one step of iterative refinement, even with single precision accumulation of residuals, guarantees such a small backward error if the matrix **A** is not too ill-conditioned and the solution components do not vary too much in magnitude.

The above theory breaks down if, for some i, b_i is zero and row i of **A** is structurally orthogonal to $\tilde{\mathbf{x}}$. Such an occurrence can be quite common if **A** is a sparse matrix. Arioli, Demmel and Duff (1988) overcome this difficulty by allowing larger relative perturbations of **b** for selected components. At each step of iterative refinement they partition the equations into two categories according to whether $(|\mathbf{A}|\,|\tilde{\mathbf{x}}|+|\mathbf{b}|)_i$ is greater than a threshold $\tau_i = 1000n\varepsilon(\sum_{j=1}^{n}|a_{ij}|\,\|\tilde{\mathbf{x}}\|_\infty + b_i)$ (category 1) or not (category 2), where ε is the machine precision. Then they compute

$$\omega_1 \equiv \max_j \frac{|\mathbf{A}^{(1)}\,\tilde{\mathbf{x}}-\mathbf{b}^{(1)}|_j}{(|\mathbf{A}^{(1)}|\,|\tilde{\mathbf{x}}|+|\mathbf{b}^{(1)}|)_j}, \text{ and } \omega_2 \equiv \max_j \frac{|\mathbf{A}^{(2)}\,\tilde{\mathbf{x}}-\mathbf{b}^{(2)}|_j}{(|\mathbf{A}^{(2)}|\,[\,|\tilde{\mathbf{x}}|+\mathbf{e}\|\tilde{\mathbf{x}}\|_\infty\,]\,)_j},$$

where **e** is the column vector of all ones. The rows of **A** corresponding to the two categories are denoted by $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ respectively, and the components of **b** are denoted similarly.

The iterative refinement stops when $\omega_1+\omega_2$ is less than ε or the value does not decrease by at least an assigned factor. Thus we have that the computed solution is the exact solution of $(\mathbf{A}+\delta\mathbf{A})\,\tilde{\mathbf{x}} = \mathbf{b}+\delta\mathbf{b}$ with

$$|\delta\mathbf{A}| \le \begin{pmatrix} \omega_1|\mathbf{A}^{(1)}| \\ \omega_2|\mathbf{A}^{(2)}| \end{pmatrix} \qquad \text{and} \qquad |\delta\mathbf{b}| \le \begin{pmatrix} \omega_1|\mathbf{b}^{(1)}| \\ \omega_2|\mathbf{A}^{(2)}|(|\tilde{\mathbf{x}}|+\mathbf{e}\|\tilde{\mathbf{x}}\|_\infty) \end{pmatrix}.$$

Arioli *et al.* (1988) also discuss a condition estimator corresponding to this new backward error which provides the values of the condition numbers κ_{ω_1} and κ_{ω_2} corresponding to ω_1 and ω_2 so that the ∞–norm of the error is bounded by

$$\frac{\|\tilde{\mathbf{x}}-\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \leq \omega_1\ \kappa_{\omega_1}+\omega_2\ \kappa_{\omega_2}.$$

This error estimate is generally tighter than estimates provided by standard condition estimators.

The method used to estimate the condition numbers is based on that developed by Hager (1984). This estimator is implemented using the subroutine `MC71A/AD`.

References

Arioli, M., Demmel J. W., and Duff, I. S. (1988). Solving sparse systems with sparse backward error. Report CSS 214, CSS Division, Harwell Laboratory, England.

Oettli, W. and Prager, W. (1964). Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik* **6**, 405-409.

Hager, W. W. (1984). Condition Estimates. *SIAM J. Sci. Stat. Comput.* **5**, 311-316.

Skeel, R. D. (1980). Iterative refinement implies numerical stability for Gaussian elimination. *Mathematics of Computation* **35**, 817-832.

5 EXAMPLE OF USE

The example code of Figure 1 shows the use of the subroutine `MA60AD` when the subroutine `MA48` is used to solve the sparse system

$$\begin{pmatrix} 0.33\dot{3}\,10^9 & 10^{-11} & & & \\ 1 & 0.33\dot{3}\,10^9 & 10^{-11} & & \\ & 1 & 0.33\dot{3}\,10^9 & 10^{-11} & \\ & & 1 & 1 & 0.33\dot{3}\,10^9 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 0.33\dot{3}\,10^9 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

When the program of Figure 1 is run on the input

```
    4   11
    1    1    2    2    2    3    3    3    4    4    4
    1    2    1    2    3    2    3    4    2    3    4
0.33333333333333D+09 0.10000000000000D-10 0.10000000000000D+01
0.33333333333333D+09 0.10000000000000D-10 0.10000000000000D+01
0.33333333333333D+09 0.10000000000000D-10 0.10000000000000D+01
0.10000000000000D+01 0.33333333333333D+09
0.33333333333333D+09 0.10000000000000D+01 0.00000000000000D+00
0.00000000000000D+00
```

it produces the output

```
 NOITER            0
 OMEGA(1)   0.000D+00
 OMEGA(2)   0.000D+00
 ERX        0.000D+00
```

```
      INTEGER MAXN,MAXNZ,LA
      PARAMETER (MAXN=100,MAXNZ=3000,LA=20000)
      INTEGER IRN(LA),ICN(LA),IRNORG(MAXNZ),ICNORG(MAXNZ),
     +   IKEEP(10*MAXN+7),IW2(2*MAXN),IW(9*MAXN),JOB(2),NOITER,
     +   KEEP(10),ICNTL(5),ICNT48(20),INFO48(20)
      DOUBLE PRECISION A(LA),AORG(MAXNZ),RHS(MAXN),X(MAXN),R(MAXN),
     +  WW(3*MAXN),W(4*MAXN),D(MAXN),RKEEP(10),CNTL48(10),
     +  ERROR(3),RX(MAXN)
      DOUBLE PRECISION OMEGA(2),COND(2),ERX,ONE,RINF48(10)
      INTEGER N,NZ,I,K,KASE,KKKK
      DATA ONE/1.0D0/
C
C       READ SPARSE MATRIX
      READ(5,*) N, NZ
      READ(5,*) (IRN(I),I=1,NZ)
      READ(5,*) (ICN(I),I=1,NZ)
      READ(5,*) (A(I),I=1,NZ)
      READ(5,*) (RHS(I),I=1,N)
      DO 30 K = 1,N
        D(K) = ONE
   30 CONTINUE
C
C       COPY THE MATRIX
      DO 40 K = 1,NZ
        IRNORG(K) = IRN(K)
        ICNORG(K) = ICN(K)
        AORG(K) = A(K)
   40 CONTINUE
C
C       FACTORIZATION
      CALL MA48ID(CNTL48,ICNT48)
      CALL MA48AD(N,N,NZ,1,LA,A,IRN,ICN,IKEEP,CNTL48,ICNT48,IW,INFO48,
     +           RINF48)
      CALL MA48BD(N,N,NZ,1,LA,A,IRN,ICN,IKEEP,CNTL48,ICNT48,W,IW,
     +           INFO48,RINF48)
C
C       ITERATIVE REFINEMENT AND ERROR ESTIMATE
C
      CALL MA60ID(ICNTL,KEEP,RKEEP)
      CALL MA48CD(N,N,.FALSE.,1,LA,A,IRN,IKEEP,CNTL48,ICNT48,RHS,X,
     +           ERROR,W,IW,INFO48)
      KASE = 0
      JOB(1) = 1
      JOB(2) = 0
      DO 100 KKKK = 1,50
        CALL MA60AD(N,NZ,AORG,IRNORG,ICNORG,RHS,X,R,D,WW,IW2,KASE,OMEGA,
     +              ERX,JOB,COND,NOITER,ICNTL,KEEP,RKEEP)
        IF (KASE.LT.0) GO TO 999
        IF (KASE.EQ.0) GO TO 888
        CALL MA48CD(N,N,KASE.EQ.1,1,LA,A,IRN,IKEEP,CNTL48,ICNT48,R,RX,
     +             ERROR,W,IW,INFO48)
        DO 50 I = 1,N
          R(I) = RX(I)
   50   CONTINUE
  100 CONTINUE
  888 CONTINUE
C
      WRITE( 6, '(1X,A10,I10)' ) 'NOITER    ', NOITER
      WRITE( 6, '(1X,A10,1P,D10.3)' ) 'OMEGA(1)  ', OMEGA(1)
      WRITE( 6, '(1X,A10,1P,D10.3)' ) 'OMEGA(2)  ', OMEGA(2)
      WRITE( 6, '(1X,A10,1P,D10.3)' ) 'ERX       ', ERX
C
      STOP

  999 WRITE (6, '(1X,A,1X,I3)') ' ERROR CODE = ',KASE
      END
```

Figure 1. Code to solve a sparse system and to estimate the error.