MA20

1 SUMMARY

To calculate **an L_1 solution to an over-determined system of m linear equations in n unknowns,** that is given equations

$$\sum_{j=1}^{n} a_{ij}\, x_j = b_i \quad i=1,2,...,m, m \geq n$$

calculate the solution vector $\mathbf{x} = \{x_j\}$ such that the sum of absolute values of the residuals

$$F(\mathbf{x}) = \sum_{i=1}^{m} |b_i - \sum_{j=1}^{n} a_{ij}\, x_j| \tag{1}$$

is minimized. A second version of the routine is provided in which (1) is minimized subject to the constraints $x_j \geq 0$, for j=1,2,...,n.

The routine can be used to solve the linear L_1 **data fitting problem;** see section 5 for an example. If the data contain some “wild” points (i.e. values of the dependent variable that are wildly inaccurate to the overall accuracy of the data) it is advisable to calculate an L_1 approximation rather than an L_2 (least-squares) approximation.

Both versions of this routine use a modification of the standard form of the simplex method applied to the primal formulation of the L_1 problem as a linear programming problem.

ATTRIBUTES — Version: 1.0.0. **Types:** `MA20A, MA20B, MA20AD, MA20BD.` **Calls:** `FD05.` **Original date:** June 1972. **Origin:** I. Barrodale, University of Victoria, Canada.

2 HOW TO USE THE PACKAGE

To calculate an L_1 solution, i.e. to determine real numbers x_j that minimize (1)

The single precision version

```
CALL MA20A(Q,D,A,R,S,IQ,M,N,TOLER)
```

The double precision version

```
CALL MA20AD(Q,D,A,R,S,IQ,M,N,TOLER)
```

To calculate a non-negative L_1 solution, i.e. to determine real numbers $x_j \geq 0$ that minimize (1),

The single precision version

```
CALL MA20B(Q,D,A,R,S,IQ,M,N,TOLER)
```

The double precision version

```
CALL MA20BD(Q,D,A,R,S,IQ,M,N,TOLER)
```

The argument lists of both versions of this routine are identical; both versions require the same input information. The output is almost identical in both versions, the only difference being that `MA20A/AD` supplies two extra pieces of information concerning the L_1 solution (see (ii) and (iii) below).

`Q` is a `REAL` (`DOUBLE PRECISION` in the D version) array of dimensions (m+2,n+2). The user must set the first m rows and n columns of `Q` to the coefficients a_{ij} of the equations, i.e. set `Q(I,J)` = a_{ij} for i=1,2,...,m, j=1,2,...,n. In addition, the first m rows of the (n+1)th column must be set to the right-hand sides b_i, i.e. set `Q(I,N+1)` = b_i for i=1,2,...,m. No other input is required to `Q.` The contents of `Q` are overwritten and, upon return, the following information is stored in `Q:`

(i) `Q(M+1,n+1)` returns the minimum value of (1), i.e. the minimum sum of absolute values of the residuals.

(ii) `Q(M+1,n+2)` returns the rank of the $m \times n$ matrix $\{a_{ij}\}$ of coefficients. If the rank is less than n, the surplus columns of the matrix $\{a_{ij}\}$ are stored in the first columns of `Q.` This information is **not** available in `MA20B,` although neither version of this routine requires the matrix $\{a_{ij}\}$ to be of full rank n.

(iii) `Q(m+2,n+1)` returns the value 1 if a solution has been calculated successfully, and the value 2 if the calculations have been terminated prematurely. This latter condition occurs only when a pivot is encountered which is almost zero, and in this event all output information pertains to the last iteration of the simplex method that was completed. In `MA20AD/AD` the value 1 **also** indicates that the solution is unique, while the alternative value 0 means that the solution is almost certainly not unique. (This uncertainty can only be resolved by a close examination of the final simplex tableau contained within `Q:` we do not consider such an examination to be warranted in practice). A solution may be non-unique simply because the matrix $\{a_{ij}\}$ is not of full rank.

(iv) `Q(m+2,n+2)` returns the number of iterations required by the simplex method.

`D` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length m in which the routine will return the residuals $(b_i - \sum_{j=1}^{n} a_{ij} x_j)$ for i=1,2,...,m.

`A` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length n in which the routine will return the solution $x_1, x_2, ..., x_n$.

`R` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length m which the user must provide for the routine to use as workspace.

`S` is an `INTEGER` array of length m which the user must provide for the routine to use as workspace.

`IQ` is an `INTEGER` variable which the user must set to the first dimension of `Q.`

`M` is an `INTEGER` variable which the user must set to m, the number of equations.

`N` is an `INTEGER` variable which the user must set to n, the number of unknowns (parameters).

`TOLER` is a `REAL` (`DOUBLE PRECISION` in the D version) variable which the user must set to a small positive value. Essentially, both versions of this routine regard any quantity as zero unless its magnitude exceeds the value of `TOLER.` In particular, the routine will not pivot on any number whose magnitude is less than or equal to `TOLER.` However, an initial inspection of the range and accuracy of the coefficients a_{ij} might suggest a more appropriate value for a given problem.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided by the user in `R` and `S,` a total of 2m words.

Input/output: None.

Restrictions: None

4 METHOD

The standard form of the simplex method of linear programming is applied to the primal formulation of the L_1 problem. A feature of the routine is its ability to pass through several simplex vertices at each iteration. The method is discussed in *An improved algorithm for discrete L_1 linear approximation* by I. Barrodale and F.D.K. Roberts, Mathematics Department Preprint, University of Victoria, Victoria, B.C., Canada.

The non-negative L_1 solution is calculated using basically the same method, except that the variables x_j are not first expressed as the difference of two non-negative variables.

5. EXAMPLE

A typical application is that of fitting a function, which depends linearly upon its parameters x_j, to a set of data which probably contains a few quite large errors.

Suppose the data is composed of m points (t_i,y_i), and that we wish to determine a polynomial in t of degree n–1 which best fits this data. Then, provided that we define a “best fit” in the L_1 sense, this problem is equivalent to finding an L_1 solution to the following over-determined system of linear equations:

$$
\begin{array}{l}
x_1 + t_1 x_2 + t_1^2 x_3 + \cdots\cdots + t_1^{n-1} x_n = y_1 \\
x_1 + t_2 x_2 + t_2^2 x_3 + \cdots\cdots + t_2^{n-1} x_n = y_2 \\
\qquad\qquad . \; . \; . \; . \; . \\
\qquad\qquad . \; . \; . \; . \; . \\
\qquad\qquad . \; . \; . \; . \; . \\
x_1 + t_m x_2 + t_m^2 x_3 + \cdots\cdots + t_m^{n-1} x_n = y_m
\end{array}
$$

Setting up this problem for either version of the routine can be accomplished by reference to section 2, provided that we make the substitutions $a_{ij} \equiv t_i^{j-1}$ and $b_i \equiv y_i$.