1 SUMMARY

This package uses the simplex method to solve the **linear programming problem**

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} = \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

subject to the constraints

$$\mathbf{Ax} = \mathbf{b}, \quad \text{that is,} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i, \ i = 1, 2, \ldots m \tag{1.2}$$

$$l_j \le x_j \le u_j, \ 1 \le j \le k, \tag{1.3}$$

$$x_j \ge 0, \ l \le j \le n. \tag{1.4}$$

The variables x_j, $k+1 \le j \le l-1$, if any, are free (have no bounds). Full advantage is taken of any zero coefficients a_{ij}. The inequalities $0 \le k < l \le n+1$ must hold. Special values $l_i = -\sigma$ and $u_i = \sigma$ may be used in (1.3) to remove one or both bounds (see Section 2.2).

The problem is solved by dividing the variables x_i into two sets; m basic variables and $n-m$ non-basic variables. The non-basic variables have values equal to zero or one of the bounds (1.3) and the basic variables are then determined by (1.2). The simplex method adjusts these sets automatically until sets that define an optimal solution are obtained. Each iteration of the method consists of interchanging the roles of a basic variable and a non-basic variable, or changing the value of a non-basic variable. The values of the basic variables are determined by solving systems of linear equations each of whose coefficient matrix, the basis matrix, is made up of the columns of **A** corresponding to basic variables.

The variable that becomes basic is chosen by finding the non-basic variable that gives the best rate of improvement of the objective function (1.1) when it is moved away from its bound and the basic variables are altered to maintain the equalities (1.2). This is achieved by computing the reduced costs (each is the rate of change of the objective function with respect to a change in a variable moving off its bound) and scaling by the steepest-edge weights (each is the corresponding rate of change of the Euclidean norm of the changes to all the variables).

To allow for the possibility of inconsistencies in the constraints (1.2) and to enable an initial choice of basis to be made easily, artificial variables x_{n+i} ($i = 1, 2, \ldots, m$) are added, with the constraints

$$x_{n+i} = 0, \quad i = 1, 2, \ldots, m \tag{1.5}$$

and the equations (1.2) are replaced by

$$\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i, \quad i = 1, 2, \ldots m, \tag{1.6}$$

The main computation actually proceeds in two phases. In the first phase, some of the variables lie outside their bounds and the objective function is replaced by the sum of these infeasibilities. If this is reduced to zero, a second phase is commenced and a sequence of feasible solutions to (1.2) – (1.4) are generated until one that solves (1.1) is obtained. Further details are given in Section 4.

To accommodate roundoff, all variables are permitted to lie slightly outside their bounds.

ATTRIBUTES — **Version:** 1.2.0. (31 January 2011) **Types:** Real (single, double). **Calls:** `FA14` `LA15`, `MC29`. **Helpful:** `MC59` (sorting). **Original date:** March 1999. **Origin:** J. K. Reid, Rutherford Appleton Laboratory. **Remark:** From May 2001 `LA04` has been made threadsafe (a small increase in workspace is now required).

2 HOW TO USE THE PACKAGE

2.1 Precision

Although there are both single and double precision versions of the routine available, the user is **strongly advised** to use the double precision version unless single precision on his or her machine actually means 8-byte arithmetic.

2.2 Specification of bounds

If (1.3) is used for any variable that is bounded on one side or free, a special value σ passed in `CNTL(1)` must be used.

Note that all bounds can be expressed in the form (1.3). This is clearly convenient for the user, as the variables may then be ordered as the user prefers, but may be wasteful in that storage will be required for the $2n$ values l_i and u_i and it may execute a little slower. On the other hand, it is easy to express a problem that has all bounds of the form (1.4). When the bounds are a mixture of forms (1.3) and (1.4), some reordering of the variables may be necessary to store the bounds economically.

2.3 Sorting

To enable efficient working, the user is required to order the matrix entries by columns. This may be achieved conveniently by calling `MC59A/AD`, as illustrated by the example in Section 5. `MC59A/AD` also performs some checks on the indices.

2.4 Scaling

It is recommended that the problem be scaled in order to reduce the effects of rounding errors on the computation. To enable this task to be performed conveniently, the `JOB` = 0 entry is available. This returns column scaling factors in `WS(1:N+1)` and row scaling factors in `WS(N+2:M+N+2)` and scales the arrays thus:

```
      DO 30 J = 1,N+1
         CS(J) =  WS(J)
   30 CONTINUE

      DO 40 I = 1,M+1
         RS(I) =  WS(N+1+I)
   40 CONTINUE

      DO 50 I = 1,M
         B(I) =  B(I)*RS(I)*CS(N+1)
   50 CONTINUE

      DO 60 J = 1,N
         C(J) =  C(J)*CS(J)*RS(M+1)
   60 CONTINUE

      DO 70 I = 1,KB
         IF(ABS(BND(1,I)).NE.SIGMA) THEN
            BND(1,I) = BND(1,I)*CS(N+1)/CS(I)
         END IF
         IF(ABS(BND(2,I)).NE.SIGMA) THEN
            BND(2,I) = BND(2,I)*CS(N+1)/CS(I)
         END IF
   70 CONTINUE

      DO 80 K = 1,NNZ
         A(K) =  A(K)*RS(IRN(K))*CS(JCN(K))
   80 CONTINUE
```

This ensures that the objective function is not changed, but the solution vector x is scaled. The example in Section 5 shows code to recover the original x.

2.5 Overall control

The subroutine uses 'reverse communication'. It performs just a single iteration of the simplex method and must therefore be called repeatedly by the user under the control of the argument JOB. The advantage of this form of call is that it gives the user more control. For instance, the user can provide his or her own monitoring printing and make his or her own decisions about dumping for restarting later. For a simple example of how the subroutine is called repeatedly to solve a problem, see Section 5 and for details about restarting, see Section 2.11.

The array argument CNTL is used to control the action of the subroutine. Default values for its elements should be set by calling the subroutine LA04I/ID before the first call to LA04A/AD.

2.6 Roundoff

Even if the correct basis has been identified, the best that can be expected is that we obtain a solution of a nearby problem. We extend this concept to the identification of the basis, too, and accept a basis that is optimal for a nearby problem. This is controlled by the parameter CNTL(2). A problem with relative changes in the entries of **A**, **b** and **c** that are less than CNTL(2) is regarded as 'nearby'.

We also tolerate variables lying slightly outside their bounds. The tolerance is held in CNTL(3). It is initialized to zero by LA04I/ID and may be increased by LA04A/AD whenever a fresh computation of the solution is made, which includes the entries with JOB = 1, 2, 4, or 5. The total number of iterations may sometimes be reduced by starting with a large value of CNTL(3) and reducing it once a phase one is complete (RINFO(31) = 2). We recommend resetting CNTL(3) prior to re-entry with JOB = 1 or 2.

For safety, we recommend that the first return with JOB having the value 0, –1, or –2 be followed by re-entry with X reset to zero, JOB = 5, CNTL(3) = 0 and CNTL(5) = 1; this will cause a fresh computation of the solution and reduced costs. It may cause further iterations to occur, with iterative refinement.

2.7 Argument lists and calling sequences

2.7.1 To set default values of controlling parameters

The single precision version

```
CALL LA04I(CNTL)
```

The double precision version

```
CALL LA04ID(CNTL)
```

CNTL is a REAL (DOUBLE PRECISION in the D version) array of length 15 that need not be set by the user. On return it contains the default values. For further information, see Section 2.8.

2.7.2 To scale the problem or perform a simplex iteration

The single precision version:

```
CALL LA04A(A,LA,IRN,IP,M,N,B,C,BND,KB,LB,JOB,CNTL,IX,JX,X,Z,G,RINFO,WS,LWS,IWS,LIWS)
```

The double precision version:

```
CALL LA04AD(A,LA,IRN,IP,M,N,B,C,BND,KB,LB,JOB,CNTL,IX,JX,X,Z,G,RINFO,WS,LWS,IWS,LIWS)
```

The leading arguments A, LA, ... , LB specify the problem and must always be set by the user. These arguments are not altered except on a JOB = 0 entry, when A, B, C, and BND are scaled.

A is a REAL (DOUBLE PRECISION in the D version) array of length LA. It must be set to contain the entries of the matrix **A**, held by columns. There must not be more than one entry for a single matrix position. A column may be zero. The entries may be in any order within each column. For example, $a_{41}, a_{11}, a_{71}, a_{23}, a_{43}, a_{45}, a_{15}$, ... is suitable. A is not altered unless JOB = 0, in which case the entries are scaled (see Section 2.4) and additional entries are added at the end for the nonzeros of the vectors **b** and **c**.

LA is an INTEGER variable that must be set to the length of the array A. There must be always be room for the entries of **A**. On an entry with JOB = 0, there must also be room for the nonzeros of the vectors **b** and **c**. LA is not altered.

IRN is an INTEGER array of length LA. It must be set to contain the row indices of the nonzero entries held in A. For the example given above, it must contain 4, 1, 7, 2, 4, 4, 1, ... IRN is not altered unless JOB = 0, in which case additional entries are added at the end for the nonzeros of the vectors **b** and **c**.

IP is an INTEGER array of length N+1. IP(j), $1 \leq j \leq n$, must hold the location in A of the first entry of column j of **A**. If column j has no entries, IP(j) must contain the location in A of the first entry of the next column with entries. In the above example, IP = 1, 4, 4, 6, 6, ... IP(n+1) must hold the first unused location. IP is not altered. **Restriction:** IP(1)=1, IP(j+1)≥IP(j), $1 \leq j \leq n$.

M is an INTEGER variable that must be set to the number of equality constraints (1.2). M is not altered. **Restriction:** M ≥ 0.

N is an INTEGER variable that must be set to the number of variables. N is not altered. **Restriction:** N ≥ 0.

B is a REAL (DOUBLE PRECISION in the D version) array of length M. It must be set to hold the vector **b** and is not altered unless JOB = 0, in which case the entries are scaled (see Section 2.4).

C is a REAL (DOUBLE PRECISION in the D version) array of length N. It must be set to hold the vector **c** and is not altered unless JOB = 0, in which case the entries are scaled (see Section 2.4).

BND is a REAL (DOUBLE PRECISION in the D version) array. If KB ≥ 1, it must have dimensions (2,KB) and be set so that BND(1,j) and BND(2,j) contain l_j and u_j. The value $-\sigma$ (see Section 2.2) should be used for l_j if there is no lower bound. Likewise, the value σ should be used for u_j if there is no upper bound. BND is not referenced if KB=0. BND is not altered unless JOB = 0, in which case the entries are scaled (see Section 2.4).
Restriction: $-\sigma \leq$ BND(1,j) ≤ BND(2,j) ≤ σ, $1 \leq j \leq$ KB.

KB is an INTEGER variable that must be set to k, the number of explicit bounds (1.3). KB is not altered.
Restriction: KB ≥ 0.

LB is an INTEGER variable that must be set to l, the index of the first variable subject to inequality (1.4). LB is not altered. **Restriction:** KB < LB ≤ N+1.

JOB is an INTEGER variable that that controls the action of the subroutine. On entry, JOB should have one of the following values :

- 0 Scale the problem.
- 1 Normal initial entry.
- 2 Initial entry with basis or partial basis specified in IX and values of non-basic variables with two bounds specified through the values of JX(j), $j \leq k$.
- 3 Normal re-entry with all arguments unchanged since the last exit.
- 4 Re-entry with a request for the refactorization of the basis matrix and recomputation of the basic variables and the reduced costs. All arguments except B, C, BND, JOB, CNTL, X, Z, WS, and IWS must be unchanged since the last exit. Within BND, any elements with value σ or $-\sigma$ must be unchanged.
- 5 Re-entry with a request for recomputation of the basic variables and the reduced costs. All arguments except B, C, BND, JOB, CNTL, X, and Z must be unchanged since the last exit. Within BND, any elements with value σ or $-\sigma$ must be unchanged.
- 6,7 Special entries to obtain $\mathbf{B}^{-1}\mathbf{w}$ and $\mathbf{B}^{-T}\mathbf{w}$ respectively for a vector **w** where **B** is the current basis matrix. The vector **w** is input as the first M elements of WS. The values of the remaining elements of WS, the array IWS, and the scalars M, LWS, and RINFO(3) must be unchanged since the last exit.

On exit, JOB has one of the following values :

0 Termination with an optimal solution.

1 Successful scaling.

3 Normal exit, ready for normal re-entry.

4 Normal exit, with a request for re-entry with refactorization of the basis.

≥6 Normal exit, following entry with JOB ≥ 6.

–1 Termination because the constraints have no solution.

–2 Termination because the objective function is unbounded. The direction that leads to an unbounded solution is available in WS.

–3 The bounds on M, N, KB, and LB are not all satisfied.

–4 LWS or LIWS is too small. They must be at least IB+3*M+4 and 2*IB+10*M+12, respectively, where IB = RINFO(35).

–5 The inequalities $-\sigma \leq$ BND(1,j) $\leq$ BND(2,j) $\leq \sigma$ are not satisfied. The index j is returned in RINFO(35).

–6 The value of IP(j) is not valid. The index j is returned in RINFO(35).

–7 The value of IRN(k) is not valid. The index k is returned in RINFO(35).

–8 IRN(k) is a duplicate of a previous entry. The index k is returned in RINFO(35).

–9 JOB was outside the range 0 ≤ JOB ≤ 7 on entry.

–10 LA is too small. It must be at least RINFO(35).

–11 The inequalities 1 ≤ IX(i) ≤ 3(M+N) are not satisfied. The index i is returned in RINFO(35).

–12 The inequalities –1 ≤ JX(i) ≤ 2 are not satisfied, or JX(i) = 1 and BND(1,j) = $-\sigma$, or JX(i) = 2 and BND(2,j) = σ. The index i is returned in RINFO(35).

–13 LWS is too small. It must be at least RINFO(35).

–14 LIWS is too small. It must be at least RINFO(35).

CNTL is a REAL (DOUBLE PRECISION in the D version) array of length 15 that affects the execution, as explained in Section 2.8. CNTL(2), CNTL(3), CNTL(5), and CNTL(11) may be altered by the subroutine. The rest are not altered.

IX is an INTEGER array of length M used to specify the indices of the basic variables on a return with JOB having the value 0, 3, 4, –1, or –2. For a variable below its lower bound, n+m is added to the index. For a variable above its upper bound, 2(n+m) is added to the index. IX need not be set on initial entry with JOB = 1. It must be set on initial entry with JOB = 2; a partial basis may be specified by setting some elements of IX to zero; and if IX(i) has the value j+n+m or j+2(n+m), with $1 \leq j \leq n+m$, it is treated as having the value j. It must be unchanged on an entry with JOB having the value 3, 4, or 5. **Restriction:** 1 ≤ IX(i) ≤ 3(M+N), $1 \leq i \leq$ M.

JX is an INTEGER array of length KB used to specify which bounds are in use on a return with JOB having the value 0, 3, 4, –1, or –2. JX(j) has one of the values:

–1 Variable x_j is in the basis and feasible, with value X(j).

0 Variable x_j is out of the basis with value X(j).

1 Variable x_j has the value X(j)+BND(1,j), and we say that the lower bound l_j is 'in use'. Variable x_j may be in the basis and infeasible or out of the basis.

2 Variable x_j has the value X(j)+BND(2,j), and we say that the upper bound l_j is 'in use'. Variable x_j may be in the basis and infeasible or out of the basis.

JX need not be set on initial entry with JOB = 1. It must be set on initial entry with JOB = 2 to indicate the status of each of the first KB variables, but the corresponding values in the array X are ignored. It must be unchanged on an entry with JOB having the value 3, 4, or 5. **Restriction:** $-1 \leq$ JX(j) ≤ 2, JX(j) $\neq 1$ if BND(1,j) $= -\sigma$, JX(j) $\neq 2$ if BND(2,j) $= \sigma$, $1 \leq j \leq$ KB.

X is a REAL (DOUBLE PRECISION in the D version) array of length N+M used to pass back the values of the variables on a return with JOB having the value 0, 3, 4, –1, or –2. If the lower bound l_j is in use, JX(j) has the value 1 and x_j has the value X(j)+BND(1,j). If the upper bound u_j is in use, JX(j) has the value 2 and x_j has the value X(j)+BND(2,j). Otherwise, x_j has the value X(j). X need not be set on initial entry with JOB having the value 1 or 2. It must be unchanged on an entry with JOB having the value 3. On an entry with JOB having the value 4 or 5, the values of only the non-basic variables are taken into account; the user may make a fresh start with non-basic variables at their exact bounds by resetting the whole of X to zero.

Z is a REAL (DOUBLE PRECISION in the D version) array of length N which need never be set by the user. It is used to pass back the reduced costs for all of the non-basic variables. It must be unchanged on an entry with JOB having the value 3.

G is a REAL (DOUBLE PRECISION in the D version) array of length N which need never be set by the user. If **B** is the current basis matrix, $\mathbf{a}_j$ the j-th column of **A** and $\mathbf{y}_j = \mathbf{B}^{-1}\mathbf{a}_j$, $1 \leq j \leq n$, then G(j) is used to hold the steepest-edge scalings $1 + \mathbf{y}_j^T \mathbf{y}_j$ when the j-th variable is non-basic and is used to hold –(position in X) when the j-th variable is basic. It must be unchanged on an entry with JOB having the value 3, 4, or 5.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of length 40 which need never be set by the user. It contains information that may be of interest to the user, as detailed in Section 2.10. It must be unchanged on an entry with JOB having the value 3, 4, or 5.

WS is a REAL (DOUBLE PRECISION in the D version) array of length LWS. It is used mainly as workspace and no part of the array must be changed prior to re-entry with JOB having a value greater than 2; with the exception of the first M elements when JOB has the value 6 or 7, see the description of JOB for more details. On a return with JOB = –2, the first N elements hold the direction that leads to an unbounded solution. On a return with JOB = 1, the first N+1 elements hold the column scaling factors and the next M+1 elements hold the row scaling factors. On an entry with JOB having the value 6 or 7, the first M elements must be set by the user to hold the vector **w** on entry and on return hold the vector $\mathbf{B}^{-1}\mathbf{w}$ or $\mathbf{B}^{-T}\mathbf{w}$, where **B** is the basis; the remaining entries are not altered.

LWS is an INTEGER variable which must be set by the user to the size of WS. For an entry with JOB = 0, it must be at least 4*M+4*N+10. For other entries, it must be at least MAX(N,IB+3*M+1)+3, where IB is a limit the number of entries permitted in the factorized basis. Advice on a suitable size for IB is given in Section 2.12. LWS is not altered.

IWS is an INTEGER workspace of length LIWS. It must not be changed prior to re-entry when JOB has a value greater than 2.

LIWS is an INTEGER variable which must be set by the user to the size of IWS. For an entry with JOB = 0, it must be at least LA+11. For other entries, it must be at least 2*IB+10*M+12, where IB is a limit the number of entries permitted in the factorized basis. Advice on a suitable size for IB is given in Section 2.12. LIWS is not altered.

2.8 The array for control

The elements of the array CNTL contain parameters that control the action of LA04A/AD. Default values for the elements are set by LA04I/ID.

CNTL(1) has default value HUGE(1.0) (HUGE(1.0d0) in the D version). This is the special value σ used for infinite bounds.

CNTL(2) has default value $\varepsilon^{2/3}$ where ε is the machine precision. This expresses the tolerable relative perturbation of the entries of **A**, **b** and **c** (see Section 2.6). If CNTL(2) has a value smaller than 10ε on an entry with JOB = 1 or JOB = 2, it is reset to this value.

CNTL(3) holds a tolerance for the bounds on the variables x_i and the artificial variables. A variable that is outside a bound by less than CNTL(3) is regarded as feasible. CNTL(3) may be increased by LA04A/AD whenever a fresh computation of the solution is made, which includes the entries with JOB = 1, 2, 4, or 5. It is initialized to zero by LA04I/ID.

CNTL(4) has default value 1. If CNTL(4)=1, steepest-edge weights are employed. If CNTL(4)=0, the steepest-edge weights are not calculated and the reduced costs are not scaled.

CNTL(5) has default value 0. It controls whether iterative refinement is performed during normal iteration. Iterative refinement is performed only if CNTL(5) ≥ 1. If the number of infeasibilities increases, this is taken to indicate ill-conditioning and CNTL(5) is set to 1. CNTL(5) may be changed by the user on any call.

CNTL(6) has default value 6. INT(CNTL(6)) is the stream number for error diagnostics. If it is set negative, this printing is suppressed.

CNTL(7) has default value 6. INT(CNTL(7)) is the stream number for printing which monitors the iteration (see Section 2.10). If it is set negative, this printing is suppressed.

CNTL(8) has default value 0.1 and holds relative pivot tolerance used when finding the initial basis and the initial value for the relative pivot tolerance U in the parameter list of LA15A/AD. It controls the selection of pivot elements in the matrix factorizations. Any potential pivot which is less than U times the largest entry in its row is excluded as a candidate. Decreasing CNTL(8) biases the factorization algorithm towards maintaining sparsity at the expense of stability and vice-versa.

CNTL(9) has default value $\varepsilon^{-1/2}$ where ε is the machine precision. This is used to set limits on the growth parameter G of LA15. If CNTL(9)* max$|a_{ij}|$ is exceeded after an LA15A/AD call, the relative pivot tolerance U is reset to min(1,2*U). If CNTL(9)$^{3/2}$ * max$|a_{ij}|$ is exceeded after an LA15C/CD call, a fresh LA15A/AD call is made.

CNTL(10) has default value ε, the machine precision. It is used for the control parameter CNTL(1) of LA15. Elements of the basis with absolute value less than CNTL(10)* max $|a_{ij}|$ are treated as zero by LA15.

CNTL(11) has default value $\varepsilon^{2/3}$ where ε is the machine precision. Entries of the vector w of changes to the solution in an iteration may be treated as zero if their absolute value is less than CNTL(11)* $\|w\|_\infty$. If CNTL(11) has a value smaller than 10ε on an entry with JOB = 1 or JOB = 2, it is reset to this value.

CNTL(12) has default value zero and controls the initialization of the random number seed used by FA14. The default is for LA04 to to initialize the seed on entries when JOB = 0, 1 or 2. If CNTL(12) is any other value than zero, one say, it is assumed the user is controlling the seed – which is always held in IWS(LIWS).

CNTL(13) to CNTL(15) are not used in the current release.

2.9 The array for information

The array RINFO contains variables that need to be preserved between exit and re-entry (JOB with value 3, 4, or 5) and which the user may wish to monitor.

RINFO(1) contains the objective function if the current solution is feasible and the sum of the moduli of the infeasibilities otherwise.

RINFO(2) holds the parameter U of LA15A/AD and LA15C/CD.

RINFO(3) contains the parameter G of LA15A/AD and LA15C/CD.

RINFO(4) holds the central processor time spent in LA04A/AD and subroutines called from it since the last initial entry (JOB with value 1 or 2).

RINFO(5) holds the central processor time spent in LA15A/AD since the last initial entry.

RINFO(6) holds the central processor time spent in LA15B/BD since the last initial entry.

RINFO(7) holds the central processor time spent in LA15C/CD since the last initial entry.

RINFO(8) contains the central processor time spent on the last refactorization of the basis matrix and recalculation of X and Z.

RINFO(9) holds the central processor clock reading on the last LA04A/AD call.

RINFO(10) holds the central processor time spent in LA04A/AD and subroutines it calls between the last initial entry (JOB with value 1 or 2) and the completion of the last refactorization.

RINFO(11) to RINFO(20) are used to hold the last ten values of the average iteration time since the last refactorization.

RINFO(21) is used to hold the minimum average iteration time since the last refactorization.

RINFO(22) is not used in the current release.

RINFO(23) is set to max $|a_{ij}|$.

RINFO(24) holds the maximum permitted value for the parameter G of LA15A/AD. If this value is exceeded after an LA15C/CD call, a fresh factorization is made.

RINFO(25) holds the index of the variable leaving the basis.

RINFO(26) holds the index of the variable entering the basis.

RINFO(27) holds the total number of infeasibilities (i.e. the total number of variables outside a bound by more than CNTL(3)).

RINFO(28) holds the number of iterations since the last refactorization.

RINFO(29) holds the number of iterations since the last initial entry (JOB with value 1 or 2).

RINFO(30) holds the number of refactorizations since the last initial entry.

RINFO(31) has the value 2 if the current solution is feasible and 1 otherwise.

RINFO(32) holds the number of entries in the factorized basis.

RINFO(33) holds the number of compresses of the factorized basis since the previous refactorization.

RINFO(34) holds the number of times that the matrix **A** was swept when looking for an initial basis.

RINFO(35) holds additional information on a return with JOB ≤ -4.

RINFO(36) to RINFO(40) are not used in the current release.

2.10 Monitoring output

Monitoring printout is provided on stream CNTL(7). It may be suppressed by setting CNTL(7) to zero.

On initial entry (JOB with value 1 or 2), the values of the scalar arguments, the number of nonzeros in the problem, and the values of the elements of CNTL are printed. If no initial basis is provided (JOB = 1), LA04A/AD picks a basis that is a permutation of an upper-triangular matrix by performing a sequence of sweeps of the matrix entries and a line of output is given for each sweep, containing:

(i) The sweep number.

(ii) The number of variables introduced into the basis so far.

(iii) The number of these variables which are artificial.

(iv) The minimum number of nonzeros found in a column of the remaining matrix.

(v) The number of columns having this minimum number of nonzeros.

Before each refactorization, a message is printed explaining that this is about to happen and why. At each factorization, the value of the parameter U of LA15A/AB/AD is printed along with the total time taken so far by the subroutine, the size of the arrays given to LA15, the tolerance on the bounds (CNTL(3)), the number of infeasibilities,

and the value of the object function. Iterative refinement is applied to the calculation of the solution and the vector $\mathbf{B}^{-T}\mathbf{c}_B$, needed to calculate the reduced costs, where $\mathbf{B}$ is the basis matrix and $\mathbf{c}_B$ is the basic part of $\mathbf{c}$.

At each iteration two lines of output are produced showing :

(i) The iteration number.

(ii) The length LENL of the **L** part of the factorization of the basis matrix.

(iii) The length LENU of the **U** part of the factorization of the basis matrix.

(iv) The variable, JIN, entering the basis.

(v) The variable, JOUT, leaving the basis.

(vi) The number, NINF, of infeasibilities.

(vii) The value of the objective function (or the sum of the infeasibilities if the phase-one problem is being solved).

(viii) The growth parameter, G, of LA15A/AD.

(ix) The number, NCP, of compresses performed by LA15, i.e. (the parameter KEEP(3) in LA15).

(x) The reduced cost, Z, of the incoming variable.

(xi) The recurred approximation to this reduced cost.

(xii) The steepest-edge weight γ used to choose the incoming variable.

(xiii) The recurred approximation to this weight.

(xiv) The central processor time since the last initial entry (JOB with value 1 or 2).

(xv) The value of the pivot.

When the run terminates, a message to this effect is printed and the value of the objective function is output.

2.11 Restarting

For a huge problem, the user may wish to preserve on file sufficient information for a later restart, even if the calculation has not been completed. Each normal (JOB = 3) re-entry requires the preservation of all arguments and this may be achieved by preserving them on file and reading them on a later run. A simpler procedure, requiring less file space, is to preserve IX (the basis), JX (indicating values of non-basic variables) and G (the steepest-edge weights) and restart with a JOB = 4 entry (a fresh factorization of the basis matrix). This entry is also suitable for restarting after a failure because of insufficient workspace (JOB = –4) provided BND, IX, JX, and G are preserved and more workspace is provided.

When one problem has been solved and a very similar one is to be solved, the old basis (or the part of it which is meaningful for the new problem) may be input by entering with JOB = 2. A good initial basis can greatly reduce the number of iterations required. If the matrix is unchanged, refactorization of the basis may be avoided by entering with JOB = 5.

2.12 Workspace requirements

The workspaces are divided dynamically. Some parts are of fixed length and the rest is given to LA15A/AD to hold the factorized basis matrix. The number of nonzeros is returned in RINFO(32) and the numbers for the two factors are shown in LENL and LENU on the monitoring print out. IB must exceed this by a reasonable margin (e.g. 10%). More space will probably be advantageous if the number of compresses (RINFO(33) or NCP in the monitor printout) rises rapidly with the iteration count (e.g. rising every other iteration).

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: FA14A/AD, LA04B/BD, LA04C/CD, LA04D/DD, LA04E/ED, LA04S/SD, LA15A/AD, LA15B/BD, LA15C/CD, and MC29A/AD.

Input/output: Output is under the control of CNTL(6) and CNTL(7).

Portability: The Fortran 95 intrinsics CPU TIME, HUGE, EPSILON.

Restrictions: M ≥ 0; N ≥ 0; 0 ≤ KB < LB ≤ N+1;
IP(1)=1; IP(j+1)≥IP(j), $1 \le j \le n$;
$-\sigma \le$ BND(1,j) ≤ BND(2,j) ≤ σ, $1 \le j \le$ KB;
$1 \le$ IX(i) ≤ 3(M+N), $1 \le i \le$ M;
$-1 \le$ JX(i) ≤ 2, JX(j) ≠ 1 if BND(1,j) = $-\sigma$, JX(j) ≠ 2 if BND(2,j) = σ, $1 \le j \le$ KB.

4 METHOD

Once a selection of m–n independent (non-basic) variables is made, the constraints (1.6) determine the remaining m dependent (basic) variables. The simplex method is a scheme for systematically adjusting the choice of basic and non-basic variables until a set which defines an optimal solution of (1.1) is obtained. Each iteration of the simplex method requires the solution of a number of sets of linear equations whose coefficient matrix is the *basis matrix* **B**, made up of the columns of [**A** **I**] corresponding to the basic variables, or its transpose $\mathbf{B}^T$. As the basis matrices for consecutive iterations are closely related, it is normally advantageous to update (rather than recompute) their factorizations as the computation proceeds.

If an initial basis is not provided by the user (i.e. if JOB = 1), a set of basic variables which provide a (permuted) triangular basis matrix is found by the simple crash algorithm of Gould and Reid (Math. Prog. **45** (1989), 475-501). and initial steepest-edge weights are calculated. If an initial basis or partial basis is provided (i.e. JOB = 2), artificial variables may be introduced as replacements or additions so as to provide a non-singular basis matrix and the crude approximate value of unity is used to initialize all steepest-edge weights (they are revised and become more accurate as the iteration progresses). The JOB = 2 entry should be used for a partial basis only if it is nearly complete; if only a few columns of the basis are known, it is probably better to start again with JOB = 1.

Phases one (finding a feasible solution) and two (solving (1.1)) of the simplex method are applied, as appropriate, with the choice of entering variable as described by Goldfarb and Reid (Math. Prog. **12** (1977), 361-371) and the choice of leaving variable as proposed by Harris (Math. Prog. **5** (1973) 1-28).

Refactorizations of the basis matrix are performed whenever doing so will reduce the average iteration time or there is insufficient memory for its factors.

The reduced cost for the entering variable is computed afresh. If it is found to be of a different sign from the recurred value or more than 10% different in magnitude, a fresh computation of all the reduced costs is performed.

Details of the factorization and updating procedures are given by Reid (Math. Prog. **24** (1982) 55-69).

Iterative refinement is always used for the basic solution and for the reduced costs after each factorization of the basis matrix. It is also used for these quantities whenever they are recomputed, that is, at the end of phase 1 and on an entry with JOB = 5.

Scaling is performed when JOB = 0 by applying MC29A/AD to the bordered matrix

$$\begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{pmatrix}.$$

5 EXAMPLE OF USE

In the example code shown below, we read a small problem from unit 5 using the value 10^{70} for σ, solve it, and write the solution.

```
      DOUBLE PRECISION SIGMA
      PARAMETER (SIGMA=1.0D70)
      INTEGER MMAX,NMAX,LA,IB,LIWS,LWS
      PARAMETER (MMAX=10,NMAX=30,LA=100,IB=LA*4,LWS=IB+3*MMAX+4,
     +          LIWS=2*IB+10*NMAX+12)
      INTEGER I,IA,ITER,J,JOB,K,M,N,KB,LB,ICNTL(10),INFO(10)
      INTEGER IRN(LA),JCN(LA),IP(NMAX+1),IX(MMAX),JX(NMAX),IWS(LIWS)
      DOUBLE PRECISION A(LA),B(MMAX),C(NMAX),X(NMAX+MMAX),WS(LWS),
     +                 G(NMAX),Z(NMAX),BND(2,NMAX),CNTL(15),RINFO(40),
     +                 CS(NMAX+1)
      EXTERNAL MC59AD,LA04AD

C     Read the data
      READ (5,*) M,N,IA,KB,LB
      READ (5,*) (A(K),IRN(K),JCN(K),K=1,IA)
      READ (5,*) (B(I),I=1,M)
      READ (5,*) (C(J),J=1,N)
      IF (KB.GT.0) READ (5,*) (BND(1,J),BND(2,J),J=1,KB)

C Initialize CNTL
      CALL LA04ID(CNTL)
      CNTL(1) = SIGMA

C Sort the entries by columns
      ICNTL(1) = 0
      ICNTL(2) = 0
      ICNTL(3) = 0
      ICNTL(4) = 6
      ICNTL(5) = 6
      ICNTL(6) = 0
      CALL MC59AD(ICNTL,N,M,IA,IRN,LA,JCN,LA,A,N+1,IP,LIWS,IWS,INFO)

C Scale the problem
      JOB = 0
      CALL LA04AD(A,LA,IRN,IP,M,N,B,C,BND,KB,LB,JOB,CNTL,IX,JX,X,Z,G,
     +            RINFO,WS,LWS,IWS,LIWS)
      DO 10 I = 1,N
        CS(I) = WS(I)*WS(N+M+2)
   10 CONTINUE

C Perform simplex iterations
      JOB = 1
      DO 20 ITER = 1,1000
        CALL LA04AD(A,IA,IRN,IP,M,N,B,C,BND,KB,LB,JOB,CNTL,IX,JX,X,Z,G,
     +              RINFO,WS,LWS,IWS,LIWS)
        IF (JOB.EQ.0) GO TO 40
        IF (JOB.LT.0) GO TO 30
   20 CONTINUE

C Write warning message if unsuccessful
   30 WRITE (6,'(A,I4,A)') ' Solution not found after',ITER,
     +  ' iterations'
      WRITE (6,'(A,F10.6)') ' JOB value is',JOB
      STOP

C Add the bounds in use to X and unscale it
```

```
   40 DO 50 I = 1,KB
        IF (JX(I).EQ.1) X(I) = X(I) + BND(1,I)
        IF (JX(I).EQ.2) X(I) = X(I) + BND(2,I)
   50 CONTINUE
      DO 60 I = 1,N
        X(I) = X(I)*CS(I)
   60 CONTINUE

C Write solution
      WRITE (6,'(A,I4,A)') ' Solution found after',ITER,' iterations'
      WRITE (6,'(A,F10.6)') ' Solution value is',RINFO(1)
      WRITE (6,'(A,/(10F10.6))') ' Solution is', (X(J),J=1,N)

      END
```

If we wish to solve the problem

$$\text{minimize} \quad \sum_{j=1}^{n} x_j$$

subject to the constraints

$$\begin{pmatrix} 1.0 & 2.0 & & \\ & 5.0 & 3.0 & 4.0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 3.0 \\ 12.0 \end{pmatrix}$$

$$1.0 \leq x_1,$$

$$x_j \geq 0,\ 2 \leq j \leq 4.$$

suitable input would be

```
2 4 5 1 2
1.0 1 1
2.0 1 2
3.0 2 3
4.0 2 4
5.0 2 2

3.0 12.0
1.0 1.0 1.0 1.0 1.0
1.0 1.0D70
```

and the output is then

```
 Entry to LA04 with M=      2 N=      4 No. of non-zeros=      5
 LA=    100 KB=      1 LB=      2 JOB=      0 LWS=    434 LIWS=   1112
 CNTL=  1.000E+70  3.667E-11  0.000E+00   1.   0.   6.   6.
        1.000E-01  6.711E+07  2.220E-16  3.667E-11

 Entry to LA04 with M=      2 N=      4 No. of non-zeros=      5
 LA=      5 KB=      1 LB=      2 JOB=      1 LWS=    434 LIWS=   1112
 CNTL=  1.000E+70  3.667E-11  0.000E+00   1.   0.   6.   6.
        1.000E-01  6.711E+07  2.220E-16  3.667E-11

 LA04: Sweep      No.       No.     Min col   No. cols
               variables artificials length   of min len
         1         2         0         1         2

 LA04: Factorization of basis. Value of U used by LA15A was   0.1000
 Total times so far for LA15A/B/C are      0.000      0.000      0.000
 Size of LA15 arrays is      422
 Largest change, iterative refinement of X:  4.4848E-17

 LA04: tolerance on the bounds is  2.3E-11
```

```
LA04: number of infeasibilites:    1
Objective function:  4.666667E-01
Largest change, iterative refinement of reduced costs:  1.0346E-16

LA04:
  Iter LENL   JIN     NINF       G        Z         GAMA       Time
       LENU  JOUT  Obj. Fun     NCP    Aprox Z   Aprox GAMA    Pivot
    1     0     4      0     1.3E+00 -8.000E-01  2.383E+00      0.000
          3     1  0.000000E+00   0  -8.000E-01  2.383E+00   -8.0E-01


LA04: no candidates are available for entry into basis
Largest change, iterative refinement of X:  9.9275E-17

LA04: tolerance on the bounds is  2.3E-11

LA04: number of infeasibilites:    0
Objective function:  3.750000E+00
Largest change, iterative refinement of reduced costs:  1.1600E-16

LA04: Optimal solution found after     1 basis factorizations
Objective function =   3.750000E+00
Solution found after   2 iterations
Solution value is  3.750000
Solution is
 1.000000  1.000000  0.000000  1.750000
```