1 SUMMARY

Solves the **linear programming problem,** finds $\mathbf{x}=\{x_j\}_n$ which minimizes the linear function

$$f(\mathbf{x})=\sum_{j=1}^{n} c_j x_j$$

subject to the linear constraints

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

where $x_j \ge 0$ $i=1,2,...,n$.

The Revised Simplex method is used where an inverse of the basis is maintained and updated at each iteration. There is an option to allow the basis inverse to be *cleaned up* at the end of the calculation and the solution checked for ill-conditioning and if necessary the calculation is restarted. Another option allows the user to force specific columns of the LP tableau into the initial basis.

`LA01B` returns an error indicator and an estimate of the ill-conditioning. Several print options are offered.

ATTRIBUTES — Version: 1.0.0. **Types:** `LA01B, LA01BD.` **Calls:** `FD05`. **Original date:** January 1975. **Origin:** M.J.Hopper, Harwell.

2 HOW TO USE THE PACKAGE

2.1 The argument list and calling sequence

The single precision version:

```
CALL LA01B(N,M,L,A,B,C,X,F,IA,IPR,IND,WK,IER)
```

The double precision version:

```
CALL LA01BD(N,M,L,A,B,C,X,F,IA,IPR,IND,WK,IER)
```

`N` is an `INTEGER` variable and is set by the user to n the number of unknowns. **Restriction:** $n > 0$.

`M` is an `INTEGER` variable and is set by the user to m the number of constraints. **Restrictions:** $m \le n+l$, $m > 0$.

`L` is an `INTEGER` variable and must be set by the user to l the number of inequality constraints. **Restriction:** $0 \le l \le m$.

`A` is a two-dimensional `REAL` (`DOUBLE PRECISION` in the D version) array with first dimension `IA.` The elements of `A` must be set to the elements of the constraint matrix $\mathbf{A}=\{a_{ij}\}_{m\times n}$. The first l rows of `A` must contain the inequality constraints. This argument is not altered by the subroutine.

`B` is a `REAL` (`DOUBLE PRECISION` in the D version) array which the user must set to the right hand sides of the constraints b_i $i=1,2,...,m$. This argument is not altered by the subroutine.

`C` is a `REAL` (`DOUBLE PRECISION` in the D version) array which the user must set to the cost function coefficients c_j $j=1,2,...,n$. This argument is not altered by the subroutine.

`X` is a `REAL` (`DOUBLE PRECISION` in the D version) array which is set by the subroutine to the solution x_j $j=1,2,...,n$. With certain error conditions `X` will not be set and for other error conditions it may not contain the best possible solution, see §2.3.

`F` is a `REAL` (`DOUBLE PRECISION` in the D version) variable and is set by the subroutine to the optimum function value $f(\mathbf{x})$.

`IA` is an `INTEGER` variable and must be set by the user to the first dimension of the array `A`, e.g. if space is allocated for `A` by

```
DIMENSION A(100,40)
```

set `IA=100`.

`IPR` is an `INTEGER` variable and provides print options. It must be set by the user to one of the following values:

`IPR=0`	no printing
`IPR≥1`	final results
`IPR≥2`	cost value at each iteration
`IPR≥3`	basis variables at each phase 2 iteration
`IPR≥4`	basis variables at each phase 1 iteration

The Fortran stream number for the output is specified in `COMMON`, see §2.2.

`IND` is an `INTEGER` array of length at least m+1. It can be used to force columns of the LP tableau into the initial basis, see `NBASIS` in §2.2 for details. If `NBASIS=0`, which it is unless the user has reset it, `IND` need not be set. On return `IND` will contain the column numbers k_i i=1,2,...,m of the final basis.

`WK` is a `REAL` (`DOUBLE PRECISION` in the D version) array which the user must provide for the subroutine to use for work space. It must be at least (m+1)(m+3) words long. See §2.4 for details of the contents of `WK` on return.

`IER` is an `INTEGER` variable set by the subroutine to an error return flag. `IER≤0` signals a successful solution, see §2.3 for other details.

2.2 The COMMON area

The subroutine uses two `COMMON` areas which the user may also reference.

The single precision version

```
COMMON/LA01D/EPS,MAXINV,NBASIS,LP,LPD,TOL
COMMON/LA01G/TAG
```

The double precision version

```
COMMON/LA01DD/EPS,MAXINV,NBASIS,LP,LPD,TOL
COMMON/LA01GD/TAG
```

`EPS` is a `REAL` (`DOUBLE PRECISION` in the D version) variable (initial value = 0) and is set by the subroutine to the magnitude of the maximum marginal cost associated with a basic variable. Basic marginal costs should be zero, in practice a good result has been obtained if `EPS` is of the order of the machine precision of the computer. Larger values will indicate ill-conditioning.

`MAXINV` is an `INTEGER` variable (initial value = 1) and specifies a limit on the number of basis re-inversions that the subroutine is allowed to make. `MAXINV` can be reset by the user. If `MAXINV=0` the subroutine cannot make a final check on the solution by cleaning up the basis inversion and re-trying.

`NBASIS` is an `INTEGER` variable (initial value = 0) and can be used to force specific columns from the `LP` tableau into the initial basis. To do this set `NBASIS` to the number of columns to go in and `IND(J) J=1,NBASIS` to the tableau column numbers. Note that a slack column, associated with the i^{th} inequality say, will be column number n+i. If `NBASIS<M` the rest of the basis will be chosen from the tableau by the subroutine.

`LP` is an `INTEGER` variable (initial value = 6 for the line printer) and specifies the Fortran output stream to be used for printing results and iteration details. `LP` may be reset by the user to any valid stream number. If `LP=0` printing is suppressed even if `IPR>0`.

`LPD` is an `INTEGER` variable (initial value = 6 for the line printer) specifies the Fortran output stream number to be used for diagnostic messages. Treat as `LP`.

`TOL` is a `REAL` (`DOUBLE PRECISION` in the D version) variable (initial value 10^{-4} or 10^{-10} in the double precision version) and specifies the tolerance allowed on the error in the basis marginal costs, see `EPS`. If `EPS>TOL` an error return is made, see `IER=2`, §2.3. The user may reset `TOL` provided `TOL≥0`.

`TAG` is a `CHARACTER*8` text string (initial value `'   LA01B'` (`'  LA01BD'` double precision version)) and is used to tag the output messages. The user may reset the tag to another name but it should be right justified in the field for the best appearance.

2.3 The error returns (IER)

`IER≤0` indicates a successful return. `IER=0` indicates that the final solution required no additional re-inversions of the basis apart from the final one which is used as a check on the solution. If `IER<0`, the absolute value of `IER` gives the number of additional re-inversions needed before a final solution was obtained. This would indicate that ill-conditioning was present at some stage although the final basis may be free from it. If the user has suppressed re-inversion by setting the common variable `MAXINV=0` the subroutine cannot check for ill-conditioning in this way. It will signal a successful calculation, `IER=0`, and assume that the user will check for ill conditioning himself.

`IER=1` indicates that re-inversion was allowed but the number allowed was insufficient to reach the optimum solution. The solution returned is likely to be ill-conditioned and not the best.

`IER=2` indicates that the maximum marginal cost associated with a basis variable exceeds the tolerance level set in `TOL`, §2.2 i.e. `EPS>TOL`. This indicates ill-conditioning and although a final solution was reached it is likely to have poor accuracy.

`IER=3` no solution is returned because phase 1 of the simplex method failed to find a feasible solution, i.e. the constraints define a null region.

`IER=4` no solution is returned because the subroutine has failed to find *m* independent columns from the LP tableau to form a basis.

`IER=5` the solution to the problem is found to be unbounded, i.e. unconstrained, no solution is returned.

`IER=6` one of the arguments `N`, `M` or `L` is invalid. All three values are printed.

2.4 The contents of the workspace

On return from the subroutine the workspace will contain information which may be of some interest to the user. It is divided up into the segments detailed below. Section 4 should be consulted for the definition of some of the terms used.

Segment 1 is m+1 words long. On return with `IER≤2` or `IER=5` the first *m* words will contain the *m* marginal costs associated with the basis variables, i.e. $v_i = chat^T U \mathbf{a}_{k_i}$ i=1,2,...,m (see §4). These marginal costs should be of the order of the word precision of the computer.

Segment 2 is m+1 words long. On all returns, except `IER=4`, this segment will contain the current basis variables w_i i=1,2,...,m and $w_{m+1} = f(\mathbf{x})$.

Segment 3 is an (m+1) by (m+1) segment containing the *transpose* of the inverse **U** of the basis. If `IER=4` it will be incomplete. If phase II had begun it will contain the extended inverse $\tilde{\mathbf{U}}$ (see §4) with the objective function row in the $(m+1)^{th}$ column.

3 GENERAL INFORMATION

Use of Common: the subroutine uses a Common area `LA01D/DD`, see §2.2.

Workspace: the user provides the workspace through an argument `WK`, $(m+1)(m+3)$ words are required.

Other subroutines: the library subroutine `FD05` is called. `LA01B/BD` also calls the private subroutines `LA01C/CD` and `LA01E/ED`.

Input/Output: private options are provided and diagnostic printing is possible. These are normally printed on the line printer but may be suppressed, see `IPR` in §2.1 and `LP` and `LPD` in §2.2.

System dependence: none.

Restrictions:

$n > 0$,

$2 \le m \le n+l$,

$0 \le l \le m$.

4 METHOD

The method used is that of the ‘Revised Simplex’ method with a modification which allows the algorithm to be started with at most only one ‘artificial’ variable. In the brief description that follows we assume the reader is familiar with the basic method and concentrate more on its implementation in `LA01B/BD`.

Take the problem to be that given in the summary. The LP tableau $\hat{\mathbf{A}}=\{\hat{a}_{ij}\}_{m\times n+1}$ is defined as the extension of $\mathbf{A}=\{a_{ij}\}_{m\times n}$ by the partial identity matrix $\mathbf{I}_l$, consisting of the first l columns of $\mathbf{I}$, which are associated with the l slack variables introduced to make the inequality constraints into equalities. The first step is to attempt to select m independent columns from $\mathbf{A}$ to form the initial basis $\mathbf{H} = ahat_{k_i}$ $i=1,2,...,m$. This is done by building the inverse of the basis $\mathbf{U}=\mathbf{H}^{-1}$ a column at a time using the rank one update formula

$$\mathbf{U}_{(r)} = \mathbf{U}_{(r-1)} - \mathbf{U}_{(r-1)}(ahat_j - \mathbf{e}_r)\frac{\mathbf{u}_r^T}{\sigma}$$

where $\mathbf{u}_r^T$ is the r^{th} row of $\mathbf{U}_{(r-1)}$ and $\sigma = \mathbf{u}_r^T ahat_j$. The column $ahat_j$ to be entered in the basis is chosen by $\sigma = \max_i |\mathbf{u}_r^T ahat_i|$ in order to ensure stability. The initial inverse is $\mathbf{U}_{(0)} = \mathbf{I}$ and if the user chooses not to force columns into the basis the subroutine will put in the l slack columns, $\mathbf{I}_l$, first and start the sequence from $\mathbf{U}_{(l)} = \mathbf{I}_l$.

If the user has chosen to force columns into the basis the same formula is used but the set of possible columns is initially restricted to his set. When these are exhausted the rest of the LP tableau is included in the search.

When building the inverse matrix $\mathbf{U}$, columns that become basic are retained in the search set to aid the detection of a rank deficient tableau, because if a basic column is chosen twice it strongly indicates that the tableau does not contain m independent columns.

Note that the subroutine holds the inverse $\mathbf{U}$ in transposed form so that row operations, which are most frequent, become column operations in the Fortran code.

After the initial inverse has been built the current LP solution $\mathbf{w} = \mathbf{Ub}$ is checked against the positivity constraints $w_i \ge 0$ $i=1,2,...,m$. If these are all satisfied phase II of the simplex method is entered, otherwise we do a phase I.

To introduce the artificial variable for phase I we implicitly extend the tableau by one column

$$ahat_{n+l+1} = \mathbf{b} - \sum_{\substack{i=1\\ i\ne j}}^{m} ahat_{k_i}.$$

When this column replaces $ahat_{k_j}$ in the basis the current solution becomes $w_i = 1$ $i=1,2,...,m$. The column k_j is chosen to preserve stability and turns out to be chosen from max$|w_i|$. Note that if at least one $w_i < 0$ this cannot be zero. In phase I we attempt to iterate out the artificial variable w_{k_j} by minimizing it, this will mean that the marginal costs will be $\mathbf{e}_{k_j}\mathbf{U}ahat = \mathbf{u}_{k_j}^T ahat$. If the constraints have no feasible solution phase I will fail to remove w_{k_j} and the subroutine will error return. If w_{k_j} is removed phase II is entered.

To begin phase II the real cost function is brought into the problem by adding an extra row $-chat^T$ to the tableau. The subroutine then works with an m+1 by m+1 inverse

$$\tilde{\mathbf{U}} = \begin{pmatrix} \mathbf{U} & :ns & \mathbf{0} \\ .. & & .. \\ chat_k^T\mathbf{U} & : & 1 \end{pmatrix}$$

where $chat_k = \hat{c}_{k_i}$ i=1,2,...,m is the basis set of cost coefficients. The marginal costs will then be $chat_k^T\mathbf{U}ahat_i - \hat{c}_i$ i=1,2,...,n+l and we choose the largest of these to decide which column comes into the basis. If no marginal cost is found greater than zero, or if a column that is already basic is chosen, we have finished the LP iterations. Otherwise we choose the column to leave the basis, update the inverse and solution and continue the iterations using the normal linear programming rules.

If we have finished the iterations the basis is re-inverted and the solution checked. If we have really reached the solution this will be confirmed and a successful return to the caller is made. If ill-conditioning was present causing the basis inverse to be bad the re-inversion may show that we were not at the solution at all. In this case we carry on with the iterations either with phase I, if the cleaned up solution was found to be infeasible, or with phase II. The process continues for as many times as the subroutine is allowed to re-invert or when a proper solution is reached.

Before returning to the caller the subroutine computes the final marginal costs associated with the basis, i.e. $v_i = chat_k^T\mathbf{U}\mathbf{a}_{k_i} - \hat{c}_{k_i}$ i=1,2,...,m. Because these give some measure of the orthogonality of the basis and its inverse they are a useful guide to the condition of the basis. Theoretically they should be zero but in practice will be or order of the relative rounding error in the computer.