1 SUMMARY

This subroutine seeks a vector \( x = (x_1, x_2, \ldots, x_n) \) satisfying the \( m \) linear constraints

\[
\begin{align*}
 x_i & \geq l_i, \quad i = 1, 2, \ldots, n \quad (1) \\
 x_i & \leq u_i, \quad i = 1, 2, \ldots, n \quad (2) \\
 c_j^T x & \geq d_j, \quad j = 1, 2, \ldots, m-2n \quad (3)
\end{align*}
\]

any of which may be required to be satisfied as equalities. A point satisfying \( n \) of the constraints as equalities is called a vertex and the method moves from vertex to vertex until one satisfying all the constraints is found or until it is possible to say that no point satisfies all the constraints.

The user must specify which constraints are equalities. These must be linearly independent and are satisfied at every vertex tested. The user may also specify inequalities to be satisfied as equalities at the starting vertex; a good choice may greatly improve the speed. Otherwise an automatic choice is made by the subroutine from amongst the bounds (1) and (2). In this case the user must beware of setting both \( u_i \) and \( -l_i \) to large values (e.g. \( 10^{10} \)) which can adversely affect accuracy.


2 HOW TO USE THE PACKAGE

2.1 Argument List

The single precision version

\[
\text{CALL LA02A}(N, M, C, IC, D, BDL, BDU, X, KE, H, IH, LT)
\]

The double precision version

\[
\text{CALL LA02AD}(N, M, C, IC, D, BDL, BDU, X, KE, H, IH, LT)
\]

\( N \) is an INTEGER variable which the user must set to the number of variables. It is not altered by the subroutine. **Restriction:** \( n > 0 \).

\( M \) is an INTEGER variable which the user must set to the total number of constraints. It is not altered by the subroutine. **Restriction:** \( m \geq 2n \).

\( C \) is a two-dimensional REAL (DOUBLE PRECISION in the D version) array which the user must set so that \( C(i,j) \), \( i = 1, 2, \ldots, n \) hold the components of the constraint vector \( c_j \), for \( j = 1, \ldots, m-2n \). It is not altered by the subroutine.

\( IC \) is an INTEGER variable which the user must set to the first dimension of \( C \) in the DIMENSION statement which allocates space to \( C \). It is not altered by the subroutine. **Restriction:** \( IC \geq n \).

\( D \) is a REAL (DOUBLE PRECISION in the D version) array, of length \( m-2n \), which the user must set to hold \( d_j \), \( j = 1, 2, \ldots, m-2n \). It is not altered by the subroutine.

\( BDL \) is a REAL (DOUBLE PRECISION in the D version) array of length \( n \), which the user must set to hold \( l_i \), \( i = 1, 2, \ldots, n \).

\( BDU \) is a REAL (DOUBLE PRECISION in the D version) array of length \( n \), which the user must set to hold \( u_i \), \( i = 1, 2, \ldots, n \).

\( X \) is a REAL (DOUBLE PRECISION in the D version) array of length \( 2n+m \). The user must set the first \( n \) components to indicate which of the bounds (1) and (2) is more suitable for the initial point chosen by the subroutine. If it decides to include a bound on \( x_i \), then it uses the lower or upper bound according to which is nearer to \( X(i) \), i.e. it uses (1) if

All use is subject to licence. 
http://www.hsl.rl.ac.uk/ 1 
Documentation date: 8th February 2011
\[ |x(i) - l_i| \leq |x(i) - u_i| \]

and (2) otherwise. On successful return from the subroutine \(X(i), i=1,2,...,n\) hold the components of the solution found. The rest of \(X\) is used for workspace.

**K** is an INTEGER variable which the user must set to the number \(k\) of constraints he or she wishes to specify for the initial point to satisfy as equalities. On successful return it has the value \(n\), and otherwise it has the value 0. **Restriction:** \(0 \leq K \leq n\).

**KE** is an INTEGER variable which the user must set to the number of constraints that are equalities. **Restriction:** \(0 \leq KE \leq k\).

**H** is REAL (DOUBLE PRECISION in the D version) work array of dimensions \((IH,N+K)\).

**IH** is an INTEGER variable which the user must set to the first dimension of \(H\) in the DIMENSION statement that allocates space to it. It is not altered by the subroutine. **Restriction:** \(IH \geq n\).

**LT** is an INTEGER array of length \(2n+m\). The user must set \(LT(j), j \leq KE\), to indicate which constraints are equalities and \(LT(j), KE < j \leq K\) to indicate which inequality constraints he or she wishes to be satisfied initially as equalities. Values \(LT(j)\) are interpreted as labelling

- constraint \(i = LT(j)\) of (1) if \(1 \leq LT(j) \leq n\),
- constraint \(i = LT(j)-n\) of (2) if \(n+1 \leq LT(j) \leq 2n\),
- constraint \(i = LT(j)-2n\) of (3) if \(2n+1 \leq LT(j) \leq m\).

The rest of \(LT\) is used for workspace.

### 2.2 The common area and diagnostic messages

**The single precision version**

\[ COMMON/LA02B/LP,IFLAG \]

**The double precision version**

\[ COMMON/LA02BD/LP,IFLAG \]

**LP** is an INTEGER variable with default value 6, set by BLOCK DATA, that specifies the stream number for diagnostic messages. If the user resets \(LP \leq 0\), then no diagnostic messages are printed.

**IFLAG** is an INTEGER variable which on return may have one of the following values:

- 0 successful return
- 1 the problem has no solution
- 2 the given equality constraints are linearly dependent

### 3 GENERAL INFORMATION

**Use of common:** see section 2.2.

**Workspace:** see description of arguments \(X, H\) and \(LT\).

**Other routines called directly:** FD05 and MB01C/CD are called.

**Input/output:** Error messages are printed under the control of common variable \(LP\) (see section 2.2).

**Restrictions:**

- \(n > 0\)
- \(m \geq 2n\)
$IC \geq n$

$0 \leq KE \leq K \leq n$

$IH \geq n$

$1 \leq LT(i) \leq m, i=1,2,...K.$

4 METHOD

The initial vertex is chosen to satisfy as equalities the $k$ constraints specified by the user and $n-k$ of the bounds (1) and (2). If the $k$ constraints are linearly dependent and $k > KE$, then a fresh attempt with $K$ reduced to $KE$ is attempted. If the $KE$ equality constraints are dependent, an error return is made. The choice of bounds to make up the complete set is made to optimize the degree of independence, as described in the report 'The calculation of feasible points for linearly constrained optimization problems', AERE-R.6354, by R. Fletcher. Further details of the method may be found in this report. The speed is comparable with that for solving a set of $n$ linear equations.