LINOPT, A FORTRAN ROUTINE FOR SOLVING LINEAR PROGRAMMING PROBLEMS

BY  JOHN W. WINGATE

RESEARCH AND TECHNOLOGY DEPARTMENT

9 OCTOBER, 1981

Approved for public release, distribution unlimited.
**LINOPT, A FORTRAN ROUTINE FOR SOLVING LINEAR PROGRAMMING PROBLEMS**

**Authors:** John W. Wingate

**Performing Organization Name and Address:**
NAVAL SURFACE WEAPONS CENTER (R44)
WHITE OAK
SILVER SPRING, MD 20910

**Report Date:** 9 October 1981

**Distribution Statement:** Approved for public release, distribution unlimited

**Abstract:** This report documents a FORTRAN routine LINOPT for solving linear programming problems. Upper and lower bounds on all variables are permitted, and the dual problem includes as a special case linearly-constrained minimum $l_1$-norm problems. Basic theory, the algorithm used, input-output procedures and examples of use are included.
FOREWORD

This report documents a FORTRAN routine LINOPT for solving linear programming problems. Upper and lower bounds on all variables are permitted, and the dual problem includes as a special case linearly-constrained minimum $L_1$-norm problems. Basic theory, the algorithm used, input-output procedures and examples of use are included.

IRA M. BLATSTEIN

IRA M. BLATSTEIN
By direction
CONTENTS

Chapter                         Page
1 INTRODUCTION.......................... 5
2 PROBLEM FORMAT AND PROGRAM USE........ 7
3 SUPPORT FUNCTIONS AND DUALITY IN LINEAR PROGRAMMING....... 13
4 DUAL SIMPLEX METHOD.................. 37
5 EXAMPLES.................................. 21
6 LISTING.................................... 27

ILLUSTRATIONS

Figure                         Page
1 ILLUSTRATION OF DUAL OPTIMAL SOLUTIONS.
   \( (\mu_3^* = 0) \) ........................................ 14
2 TEST PROGRAM ......................... 26

TABLES

Table                         Page
1 FILLING IN MISSING BOUNDS .............. 9
2 CORRESPONDING VARIABLES .............. 10
CHAPTER 1

INTRODUCTION

REPORT ORGANIZATION

This report documents a FORTRAN subroutine (with associated subroutines) called LINOPT for solving linear programming (LP) problems. It is divided into several chapters. Following the INTRODUCTION is a chapter (PROBLEM FORMAT AND PROGRAM USE) explaining the types of problems which can be solved by LINOPT, some manipulations on them and correspondences with program notation. A chapter (SUPPORT FUNCTIONS AND DUALITY IN LINEAR PROGRAMMING) discusses some of the duality concepts behind the formulation and solution of LP problems. The next chapter, DUAL SIMPLEX METHOD gives some information about the algorithm used in the program. EXAMPLES and a LISTING follow.

PROBLEM FORMULATION

A rather abstract formulation of an LP problem in the following: Let X be a real vector space paired with M under the bilinear form (inner product) \((u,x) \rightarrow u \cdot x, u \in M, x \in X\), and let \(Y_i, i = 1, \ldots, m\) be similarly paired with \(A_i, i = 1, \ldots, m\). Given \(u \in M\), closed convex sets \(C_i\) in \(Y_i\), and linear transformations \(A_i : X \rightarrow Y_i, i = 1, \ldots, m\),

maximize \(u \cdot x\)
subject to \(A_i x \in C_i, i = 1, \ldots, m\).

This problem in convex programming has the following as its dual problem:

\[
\begin{align*}
\text{minimize} & \sum_{i=1}^{m} c_i^*(\lambda_i) \\
\text{subject to} & \sum_{i=1}^{m} A_i^* \lambda_i = u
\end{align*}
\]

where \(c_i^*(\lambda_i)\) is the support function of the set \(C_i\) — see Chapter 3 for more on support functions. For explanatory purposes it is sufficient to take each \(A_i\) equal to the identity, so that \(Y_i = X, A_i = M, i = 1, \ldots, m\). Chapter 3 treats this simplified version.

The program LINOPT is set up to handle constraints of the form \(u_i \cdot x \in C_i\) where \(u_i \in M\) and \(C_i\) is a nonempty closed interval. If some \(C_i\) is a bounded interval of nonzero length, the dual problem has a nonlinear objective; it is, however, convex and piecewise linear. The details are given in Chapter 2, where it is shown that the general form of the dual objective is the sum of two terms, one linear, and one a weighted \(l_1\) norm.
The algorithm used in the program is a form of the revised dual simplex algorithm modified to handle upper and lower bound constraints. The inverse matrix used is a row-basis inverse. Accordingly, the algorithm is more efficient on problems with many constraints. (On problems with fewer dependent variables than independent variables, a column-basis inverse would be smaller.) Use of a row basis has definite advantages in modifying a problem and then reoptimizing.

No new theory is involved in this program. The dual simplex algorithm can be found in standard linear programming texts.\textsuperscript{1,2} Insisting that all variables in the primal problem have both upper and lower bounds makes it trivial to find a dual-feasible point to start the algorithm, and because of the resulting asymmetry between primal and dual problems, allows us to handle directly (via the dual problem) certain piecewise linear convex minimization problems.

The results from convex analysis used in Chapter 3 can be found in greater generalization and detail in Rockafellar's book.\textsuperscript{3}

\begin{footnotesize}
\begin{enumerate}
\item Hadley, G., \textit{Linear Programming}, Addison-Wesley, Reading, 1962.
\end{enumerate}
\end{footnotesize}
INTRODUCTION

LINOPT is programmed to solve a problem maximizing (or minimizing) a linear function subject to upper and lower bounds on linear constraint functions. (The bounds are equal for an equality constraint.) The dual to this problem has a piecewise linear objective function and linear equality constraints. Missing bounds in the primal problem correspond to sign constraints on the dual variables. Such missing bounds can be handled by introducing a penalty function for the sign constraint violations in the dual problem. An even simpler and more direct interpretation is that the missing bounds can be replaced by bounds so large in magnitude that they are effectively infinite.

PRIMAL PROBLEM

Maximize $x_{k_0}$ subject to $b_k \leq x_k \leq \bar{b}_k$, $k=1,..., m+n$, where $k_0 \in \{1,..., m+n\}$ and

$$x_{n+i} = \sum_{j=1}^{n} a_{ij} x_j, \quad i=1,...,m.$$  

The objective variable $x_{k_0}$ can be expressed by:

$$x_{k_0} = \sum_{j=1}^{n} c_j x_j,$$

where

$$c_j = \begin{cases} s_{k_0} & \text{if } k_0 \leq n, \\ s_{k_0-n,j} & \text{if } k_0 > n. \end{cases}$$

The same data also define the dual problem.

DUAL PROBLEM

Minimize $\sum_{k=1}^{m+n} \max \{u_k b_k, u_k \bar{b}_k\}$ subject to

$$u_j + \sum_{i=1}^{m} u_{n+i} a_{ij} = c_j, \quad j=1,...,n.$$  

The dual objective has another form which is more likely to be recognized in an application:

$$\sum_{k=1}^{m+n} \max\{u_k b_k, u_k \bar{b}_k\} = \sum_{k=1}^{m+n} \left( s_k + \frac{\bar{s}_k}{2} \right) u_k + \sum_{k=1}^{m+n} \left( \frac{\bar{s}_k - s_k}{2} \right) |u_k|.$$
Thus the dual objective contains a linear term and a weighted $\ell_1$-norm term.

The dual variable $u_k$ can be thought of as a Lagrange multiplier for the constraint $x_k \in [b_k, b_k]$.

MISSING BOUNDS; SIGN CONSTRAINTS

The following table (TABLE 1) shows how to prepare primal problems with missing constraints or dual problems with sign constraints. M is a very large positive number. (The default value supplied by the program is $10^{100}$.)

The first line of the table gives the standard two-sided constraint assumed by the program. The other lines give the modifications for unilateral and no constraints. In the modified problem $u_k$ is always unconstrained in sign. The dual objective picks up the original linear term when the sign constraint is satisfied, and a penalty term when the sign constraint is violated.

When the program gives an optimal solution in which $x_k = \pm M$ for some $k$, the original problem has an unbounded solution. If it has a finite solution, the program will yield it, and it will not depend on $M$ (unless $M$ has been set so small that it interferes with the "legitimate" constraints). The calculations are arranged so that roundoff errors due to the disparity in magnitude between $M$ and the original data do not propagate from iteration to iteration, and appear within an iteration only if some $x_k = \pm M$.

The objective variable $x_{k0}$ is also formally a constrained variable, although generally the constraint will be $-M \leq x_{k0} \leq M$, i.e. essentially no constraint at all. Tighter bounds may at times be useful. The constraint $b_{k0} \leq x_k \leq M$ can be used to answer the question: Is $\max x_k \geq b_{k0}$? If the answer to this question is negative, the constraints are inconsistent. As soon as the inconsistency is detected, the program returns to the calling program without going on to calculate the solution completely.
<table>
<thead>
<tr>
<th>Original constraints on $x_k$</th>
<th>Original sign constraints on $u_k$</th>
<th>Original dual objective term</th>
<th>Modified constraints</th>
<th>Modified dual objective term</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_k &lt; x_k &lt; b_k$</td>
<td>$u_k$ unconstrained in sign</td>
<td>$\max{u_kb_k, u_k\bar{b}_k}$</td>
<td>$b_k &lt; x_k &lt; b_k$</td>
<td>$\max{u_kb_k, u_k\bar{b}_k}$</td>
</tr>
<tr>
<td>$x_k &lt; b_k$</td>
<td>$u_k &gt; 0$</td>
<td>$u_k\bar{b}_k$</td>
<td>$-M &lt; x_k &lt; \bar{b}_k$</td>
<td>$\max{u_k(-M), u_k\bar{b}_k}$</td>
</tr>
<tr>
<td>$x_k &gt; b_k$</td>
<td>$u_k &lt; 0$</td>
<td>$u_kb_k$</td>
<td>$b_k \leq x_k \leq M$</td>
<td>$\max{u_kb_k, u_kM}$</td>
</tr>
<tr>
<td>$x_k$ unconstrained</td>
<td>$u_k = 0$</td>
<td>0</td>
<td>$-M \leq x_k \leq M$</td>
<td>$M</td>
</tr>
</tbody>
</table>
PROGRAM NOTATION

Correspondence between the notation herein and the notation of LINOPT is given in Table 2.

<table>
<thead>
<tr>
<th>This TR</th>
<th>LINOPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>M</td>
</tr>
<tr>
<td>n</td>
<td>N</td>
</tr>
<tr>
<td>k_0</td>
<td>IOBJ</td>
</tr>
<tr>
<td>x_k</td>
<td>X(K)</td>
</tr>
<tr>
<td>u_k</td>
<td>U(K)</td>
</tr>
<tr>
<td>b_k</td>
<td>BL(K)</td>
</tr>
<tr>
<td>b_j</td>
<td>BU(K)</td>
</tr>
<tr>
<td>a_{ij}</td>
<td>A(ROW(I) + COL(J))</td>
</tr>
<tr>
<td>M</td>
<td>BIGM</td>
</tr>
</tbody>
</table>

The FORTRAN variable BIGM is included in the program for the user's convenience. It supplies a default value (which can be changed) for filling in the missing bounds. (The user must fill in all bounds, since there is no provision for keeping track of missing bounds otherwise.)

CONSTRAINT COEFFICIENT STORAGE

The constraint coefficients are referenced in an unusual but flexible way. Row and column pointer arrays ROW and COL are used to index an array A. The FORTRAN standard for array storage is by columns (a_{11}, a_{21}, a_{31},..., a_{12}, a_{22}, a_{32}, etc.) Suppose that we have a matrix A stored in an array dimensioned 10 x 20 and we wish to study a problem whose constraint coefficients form a submatrix of A, as in

\[
\begin{bmatrix}
y_5 \\
y_8
\end{bmatrix} = \begin{bmatrix}
a_{57} & a_{53} & a_{59} \\
a_{87} & a_{83} & a_{89}
\end{bmatrix} \begin{bmatrix}
z_7 \\
z_8 \\
z_9
\end{bmatrix}
\]

We can set x_1 = z_7, x_2 = z_3, x_3 = z_9, x_4 = y_5, x_5 = y_8,

ROW(1) = 5, ROW(2) = 8, 
COL(1) = 60, COL(2) = 20, COL(3) = 80.

COL(J) is set to the number of elements in the array preceding the coefficient column for X(J)—6 x 10 for the 7th column, 2 x 10 for the 3rd and 8 x 10 for the 9th. ROW(I) then picks off the appropriate entry in the column. We can even introduce (by using the LOCF function) columns extraneous to the array storing A. Consider the modified example:

\[
\begin{bmatrix}
y_5 \\
y_8
\end{bmatrix} = \begin{bmatrix}
a_{57} & a_{53} \\
a_{87} & a_{84}
\end{bmatrix} \begin{bmatrix}
z_7 \\
z_3
\end{bmatrix} + \begin{bmatrix}
b_{54} \\
b_{84}
\end{bmatrix} w_4
\]
in which \( x_3 = w_4 \). COL(3) is set by

\[
\text{COL}(3) = \text{LOCF(Bl,4)} - \text{LOCF(A(1,1))}.
\]

The columns of \( B \) must be structured the same way as the columns of \( A \) for this procedure to work. Instead of having a coefficient matrix stored in the array in the usual way we could have it transposed, perhaps as a result of starting with a dual problem. This corresponds to storage by rows (not the FORTRAN standard). A little reflection indicates that defining ROW the way COL is defined above, and COL the way ROW is defined handles this storage arrangement. Further examples are given in the program comments.

**INPUT AND OUTPUT**

Input and output variables are clearly indicated in the program comments. (The program is listed in the section titled "LISTINGS"). Arrays are passed as formal parameters. Scalars are passed by using a labelled common block /XXXLP/, which must accordingly be a common block in the calling program.

**ROUNDOFF CONTROL**

In the program there are three input variables which can be used to control roundoff error accumulations. \( \text{EPS} \) is a tolerance used in checking constraint violations. \( H \) is also used to zero out coefficients in the tableau which have small nonzero values (typically for a CDC 60-bit machine, on the order of \( 10^{-14} \)) which ought to vanish. Finally constraint violations less than \( \text{EPS} \) are eliminated from the optimal solution before returning. For small problems \( \text{EPS} = 0 \) is generally all right.

The other roundoff - controlling variables are \( \text{INVERT} \) (a logical variable) and \( \text{ITMAX} \). When \( \text{INVERT} \) is TRUE the inverse matrix corresponding to the current key \( K \) is calculated. \( \text{ITMAX} \) is a limit on the number of iterations. When this limit is reached control is returned to the calling \text{LINOPT} again with \( \text{INVERT} = \) TRUE., one may control the building of roundoff error in the inverse matrix (which otherwise is updated by column operations every iteration). (For small problems this may not be necessary.)

After obtaining a solution (or after any return from an initial call to \text{LINOPT}) \( \text{INVERT} \) can be set to FALSE for another call to \text{LINOPT}. Certain modifications to the problem data are permissible at such a time - constraints may be added, for example. These modifications are any for which the inverse matrix would be unchanged, and include the following (Note: primary indices: \( K(1), \ldots, K(N) \); secondary \( K(N + 1), \ldots, K(N + M) \) - see Chapter 4.)

**ADDING CONSTRAINTS**

\( M \) is increased, new elements to ROW are added to point to the new constraint coefficients (which, if not already defined should be stored appropriately), and new bounds added to BL and BU.
MODIFYING CONSTRAINTS

Bounds for any secondary variable can be changed. Inactive bounds for primary variables can be changed, provided the corresponding solution is also changed; e.g. if $X(K(1)) = BU(K(1))$ and $BU(K(1))$ is changed, $X(K(1))$ must be changed in the same way. Constraint coefficients for dependent variables $X(N + 1), \ldots, X(N + M)$ which are also secondary variables can be changed; (these may include the objective variable) or such constraints can be dropped, with appropriate changes to ROW, M, BL and BU. (If the constraint corresponding to ROW(I) is dropped, the simplest way to make these changes is to set $ROW(I) = ROW(M)$, $BL(N + I) = BL(N + M)$, $BU(N + I) = BL(N + M)$, and then $M = M - 1$, so that the indexing for $X(N + M)$ is changed to $X(N + I)$).

Of course, when LINOPT is recalled with INVERT = TRUE, any problem changes whatsoever are permissible.
CHAPTER 3
SUPPORT FUNCTIONS AND DUALITY IN LINEAR PROGRAMMING

The support function \( \sigma_C \) of a convex set \( C \) in \( X \) is a convex function defined on \( M \) by:
\[
\sigma_C (\mu) = \sup_{x \in C} \mu \cdot x
\]  
(1)

The support function of the empty set is \(-\infty\) everywhere. For nonempty \( C \), \( \sigma_C (\mu) > -\infty \) and may take the value \(+\infty\); in Rockafellar's terminology, it is a proper convex function.4

Many of the formulas of convex analysis can be simplified when they are restricted to convex polyhedra and convex polyhedral functions. One such is found in Corollary 16.4.1 of Rockafellar's book5, from part of which we can derive the following: Let \( C_1, \ldots, C_m \) be closed convex polyhedra with nonempty intersection \( C \) (also a convex polyhedron). Then
\[
\sigma_C (\mu) = \begin{cases} 
\min \sum_{i=1}^{m} \sigma_{C_i} (\mu_i) \\
\text{subject to } \sum_{i=1}^{m} \mu_i = \mu
\end{cases}
\]  
(2)

(The general version of the corollary is required if \( C \) is empty.) Rockafellar terms the operation in (2) "infimal convolution", since for \( m = 2 \), \( \sigma_{C_1 \cap C_2} (\mu) = \inf_\lambda (\sigma_{C_1} (\lambda) + \sigma_{C_2} (\mu - \lambda)) \) a form reminiscent of integral convolutions. Formulas (1) and (2) express \( \sigma_C (\mu) \) as the common optimal value of a pair of dual convex programming problems:

Primal Problem:

Maximize \( \mu \cdot x \)

subject to \( x \in C_i, \ i = 1, \ldots, m. \)

Dual problem:

Minimize \( \sum_{i=1}^{m} \sigma_{C_i} (\mu_i) \)

subject to \( \sum_{i=1}^{m} \mu_i = \mu. \)

5. Ibid, p. 146.
Three solution cases arise:

1. \( \partial C (\mu) = -\infty \): C is empty. The dual objective is unbounded below on the dual constraint set.

2. \(-\infty < \partial C (\mu) < +\infty \): C is not empty. The optimal value, viz \( \partial C (\mu) \), is attained in both problems.

3. \( \partial C (\mu) = +\infty \): C is not empty. The primal objective is unbounded above on the primal constraint set. The dual objective is \(+\infty\) everywhere on the dual constraint set, (i.e. \( \partial C_i (\mu_i) < \infty \), \( i = 1, \ldots, m \) \( \implies \sum_{i=1}^{m} \mu_i \neq u \).

Suppose that \( \partial C (\mu) \) is finite, and let \( x^* \) solve the primal, \( u_i^* \), \( i = 1, \ldots, m \), the dual. Then

\[
\begin{align*}
\partial C (\mu) &= \mu \cdot x^* \quad \text{(primal optimality)} \\
&= (\sum_{i=1}^{m} \mu_i^*) \cdot x^* \quad \text{(dual constraint)} \\
&= \sum_{i=1}^{m} (\mu_i^* \cdot x^*) \\
&\leq \sum_{i=1}^{m} \partial C_i (\mu_i^*) \quad \text{(primal constraints)} \\
&= \partial C (\mu) \quad \text{(dual optimality)}
\end{align*}
\]

It follows that

\[
\mu_i^* \cdot x^* = \partial C_i (\mu_i^*), \quad i = 1, \ldots, m,
\]

or that \( u_i^* \) supports \( C_i \) at \( x^* \): \( u_i^* \) is an outer normal to \( C_i \) at \( x^* \). The formula is one way of expressing complementary slackness, since if \( x^* \in \text{int} \ C_i \) then \( u_i^* = 0 \), while

\[\text{Figure 1 ILLUSTRATION OF DUAL OPTIMAL SOLUTIONS.} \quad (u_3^* = 0)\]
if \( u_i^* = 0 \), then the constraint \( x \in C_i \) can be dropped without altering the solution. The set of constraints for which \( u_i^* \neq 0 \) are active (binding) at the optimum. (See Figure 1). There may be multiple solutions, each with a different set of active constraints.

The usefulness of (2) hinges on picking the constraint sets \( C_i, i=1,...,m \), to be simple enough to permit easy evaluation of their support functions. Any convex polyhedron can be expressed as the intersection of half-spaces, and any half-space can be defined by a linear inequality. Let \( H = \{x: \mu \cdot x \leq b\} \), where \( \mu \neq 0 \). Then

\[
\sigma_H (\nu) = \begin{cases} 
ub \text{ if } \nu = u\mu \text{ and } u \geq 0, \\
+\infty \text{ otherwise.}
\end{cases}
\]

In terms entirely of hyperplane constraints the primal and dual problems become

Primal problem:

Maximize \( \mu \cdot x \)

subject to \( \mu_i \cdot x \leq b_i, i=1,...,m \)

Dual problem:

Minimize \( \sum_{i=1}^{m} u_i b_i \)

subject to \( \sum_{i=1}^{m} u_i \mu_i = u, \)

\( u_i \geq 0, i=1,...,m \).

The sign constraints on \( u_i \) avoid infinite values of the dual objective and keep it linear so that both problems consist of optimizing linear functionals subject to linear constraints. Alternatively we could omit the sign constraints and keep the formulation in terms of support functions. This pair of problems also illustrates the more general duality relationship cited in the INTRODUCTION. Let \( A_i: X \to \mathbb{R} \) be defined by \( A_i X = \mu_i \cdot X \). Then \( A_i^*: \mathbb{R} \to \mathbb{R} \) and \( A_i^* u = u\mu_i \). Moreover,

\[
\sigma_{(-\infty, b_i]} (u) = \begin{cases} 
ub_i \text{ if } u \geq 0, \\
+\infty \text{ if } u < 0.
\end{cases}
\]

Thus the problems are expressible as:

Primal problem:

Maximize \( \mu \cdot x \)

subject to \( A_i x \in (-\infty, b_i], i=1,...,m \)

Dual problem:

Minimize \( \sum_{i=1}^{m} \sigma_{(-\infty, b_i]} (u_i) \)

subject to \( \sum_{i=1}^{m} A_i^* u_i = u \)
We may have two-sided constraints such as

\[ b \leq u \cdot x \leq \bar{b} \]

(where \( b \leq \bar{b} \)) defining a set \( S \) in \( X \), which can be replaced by the pair of constraints

\[
\begin{align*}
\{ & u \cdot x \leq \bar{b}, \\
& -u \cdot x \leq -b.
\} 
\end{align*}
\]

since

\[ \sigma_S(v) = \begin{cases} \max \{u_b, u_\bar{b}\} & \text{if } v = uu, \\ +\infty & \text{otherwise,} \end{cases} \]

and \( \max \{u_b, u_\bar{b}\} \) is nonlinear in \( u \) (unless \( b = \bar{b} \)), the dual of a problem with two-sided constraints is nonlinear. Of course it is easy to relate the two-sided and one-sided versions by using (2). Thus if \( u \) is the dual variable for the constraint \( b \leq u \cdot x \leq \bar{b} \), \( u^+ \) for \( u \cdot x \leq b \) and \( u^- \) for \( -u \cdot x \leq -\bar{b} \),

then

\[ u = u^+ - u^- \quad (4) \]

while if \( b = \bar{b} \) either \( u^+ \) or \( u^- \) vanishes (at the solution), so that

\[ u^+ = \max \{u, 0\}, \quad -u^- = \min \{u, 0\}. \quad (5) \]

When \( b = \bar{b} \), only the difference \( u = u^+ - u^- \) is determined. An alternate viewpoint in this case is that \( u \) is the dual variable for the linear equality constraint \( u \cdot x = b \).

The pivoting operations of the dual simplex method can be thought of as substituting one hyperplane bounding a half-space for another, and consequently are better suited for the formulation in terms of one-sided constraints. The relations (4) and (5) and some sign bookkeeping then make it easy to apply the method to two-sided constraints. Explanations of the method without the sign manipulations are more transparent. Accordingly in the next section only one-sided constraints are considered.
CHAPTER 4

DUAL SIMPLEX METHOD

In the previous sect' n no particular coordinates were used on X. Most LP problems encountered are expressed in terms of coordinates with respect to some particular basis for X, the coordinates then forming a set of independent variables. (as indicated previously, LINOPT assumes such a formulation.) Thus, with one-sided constraints, we get a pair of problems like the following, in which we assume that the n independent variables \( x_j, j \in J \) are included in the m+n constrained variables \( x_k, k \in K \); i.e. \( J \subseteq K \).

Primal problem:

Maximize \( x_0: = \sum_{j \in J} c_j x_j \)

subject to \( x_k: = \sum_{j \in J} a_{kj} x_j \leq b_k, k \in K. \)

(Note that \( a_{kj} = \delta_{kj} \) for \( k \in J. \))

Dual problem:

Minimize \( \sum_{k \in K} u_k b_k \)

subject to \( \sum_{k \in K} u_k a_{kj} = c_j, j \in J \)

\( u_k \geq 0, k \in K. \)

Note that the n equations relating the dual variables \( u_k, k \in K \) can be written explicitly for \( u_j: \)

\( \sum_{i \in K-J} u_i a_{ij} + u_j = c_j \)

Thus in the dual problem, \( u_i, i \in K-J, \) are independent and \( u_j, j \in J, \) are dependent. Given some other subset \( J' \) of \( K \) for which \( x_j, j \in J', \) are linearly independent, we can transform the constraint relations so that \( x_j, j \in J' \) are the independent variables through which the primal problem is phrased. Given such an index set \( J \) we can define a corresponding basic solution. For \( J \) the definition of a basic solution is obtained by setting the independent variables \( x_j, j \in J, \) and \( u_i, i \in K-J, \) to their bounds and satisfying the constraint relations among the variables; \( x_k \) and \( u_k \) are the values of \( x_k, u_k \) at the basic solution.

\( x_j = b_j, j \in J \) (primary primal variables)

\( \bar{x}_j = \sum_{j \in J} a_{ij} b_j, i \in K-J \) (secondary primal variables)

\( u_j = c_j, j \in J \) (secondary dual variables)

\( \bar{u}_i = 0, i \in K-J \) (primary dual variables)
The terms primary and secondary have been introduced instead of independent and dependent because one may wish to refer to the primary variables of the problem as initially formulated as the independent variables. The split between primary and secondary depends on the set J and changes with it. The secondary primal indices are usually called basic indices in linear programming texts, since they correspond to the indices for a column basis for the constraint matrix. This terminology is a little inappropriate here, since LINOPT makes use of a row basis corresponding to the complementary set of indices - the dual basic indices in the usual description. The use of "basic" in this sense is avoided here to prevent confusion.

Furthermore, in a problem with two-sided constraints the basic indices refer to the indexing of the equivalent one-sided problem, not the indexing of the two-sided problems, so that the basic variables for J would be \(x_k\) and \(-x_k\) for \(k \in K \setminus J\) and either \(x_j\) or \(-x_j\) (but not both) for \(j \in J\). Alternatively we may retain the "primary/secondary" notation and supplement it with some way of indicating whether a primary variable is at its upper or its lower bound. (The program simply checks the solution value against the bounding values.

In a basic solution the primary variables satisfy the constraints placed on them if all primal variables satisfy the constraints, the basic solution and J are primal-feasible. If the dual constraints are satisfied, the basic solution is dual-feasible. A basic solution which is both primal- and dual-feasible is optimal. At a basic solution both primal and dual objective variables have the value \(\sum_{j \in J} c_j b_j\). The criterion for dual feasibility is simply that \(c_j \geq 0, j \in J\).

The transformation of the constraint coefficients accompanying a change from one set of primary variables to another can be performed explicitly when needed, or it can be expressed in terms of a nonsingular matrix relating the variables.

There are two ways of doing this. Let \(x_j, j \in J\) and \(x_j', j \in J'\) be two sets of primary primal variables. Set \(I = K \setminus J\), \(I' = K \setminus J'\), and let \(x_j\) be a column vector whose entries are \(x_j, j \in J\), etc. Using matrix notation the two ways can be described as follows:

1. Solution for \(x_I'\)
\[x_I' = Ax_J\] (A is \(m \times n\))

Rearrange columns to give:
\[B x_I' = R x_J'\]
with B a nonsingular \(m \times m\) submatrix of \([I -A]\). Then \(x_I' = B^{-1} R x_J'\).

The columns making up B are a basis for the space spanned by the columns of \([I -A]\).

Applied to the dual:
\[
\begin{align*}
U_I A + U_J &= C \\
U_I S + U_J D &= C \\
U_I' S D^{-1} + U_J' &= C, \text{ where } C = C D^{-1}.
\end{align*}
\]
The rows of \( D \) form a basis for the space spanned by the rows of \( [A I] \).

2. Solution for \( x_J \) and substitution:

\[
\begin{bmatrix}
X_I \\
X_J
\end{bmatrix} =
\begin{bmatrix}
A \\
I
\end{bmatrix}
X_J
\]

Rearrange rows to give:

\[
\begin{bmatrix}
X_I' \\
X_J'
\end{bmatrix} =
\begin{bmatrix}
S \\
D
\end{bmatrix}
X_J =
\begin{bmatrix}
SD^{-1} \\
I
\end{bmatrix}
X_J'
\]

(\( S \) and \( D \) are the same as above. \( B^{-1}R = SD^{-1} \).)

For the dual:

\[
\begin{bmatrix}
U_I \\
U_J
\end{bmatrix} = U_I[I -A] + [0 C]
\]

\[
\begin{bmatrix}
U_I' \\
U_J'
\end{bmatrix} = U_I[B -R] + [C'I C'J]
\]

\[
\begin{bmatrix}
U_I' \\
U_J'
\end{bmatrix} = U_I[I -B^{-1}R] + [0 C]
\]

where \( C = C'I + C'J \), \( B^{-1}R \) and is the same as before. The inverse matrix \( B^{-1} \) is the product of elementary row operations; \( D^{-1} \) is the product of elementary column operations. Either one may be used to keep track of changes. LINOPT uses \( D^{-1} \) and generates coefficients and solutions from the original constraint coefficients by:

\[
\begin{bmatrix}
X_I \\
X_J
\end{bmatrix} =
\begin{bmatrix}
A \\
I
\end{bmatrix}
D^{-1}X_J'
\]

When a basic solution is changed, \( D^{-1} \) is updated by column operations. (We have ignored \( x_0 \): assume that \( c \in K \) so that \( c \) is a row of \( [A I] \).)

The dual simplex algorithm works with dual-feasible basic solutions. Given a set \( J \) defining primary primal variables, \( J \) is altered by adding an index not in \( J \) and dropping an index in \( J \): a secondary variable replaces a primary variable. The resulting changes in the constraint coefficients can be accomplished by Gauss-Jordan pivoting. The indices entering and leaving \( J \) are chosen in such a way that dual-feasibility is maintained and the objective function value does not increase. Proof of convergence can be found in any linear programming text.\(^6\) The procedure is:

1. Identify violated constraints: \( I' = \{ i \in K \sim J: X_i > b_i \} \).
2. If \( I' \) is empty, stop: basic solution is optimal.
3. Pick (by some heuristic) \( k \in I' \).
4. Identify constraints which can be dropped without being violated when \( x_k \) is set to \( b_k \): \( J' = \{ j \in J: a_{kj} > 0 \} \)
5. If \( J' \) is empty, stop: constraints are inconsistent.

\(^6\) For example, Hadley, op. cit, or Simmonard, op. cit.
6. Identify subset of \( J' \) corresponding to constraints for which dual feasibility is maintained when dropped: \( J'' = \{ j \in J' : \frac{c_j}{a_{kj}} = \min_{j \in J} \left( \frac{c_j}{a_{kj}} \right) \} \).

7. Pick (while applying anticycling criterion, if desired) \( k \in J'' \).

8. Update solution: \( J = J \cup \{ k \} \sim \{ l \} \). Update inverse \( D^{-1} \) by column operations. Calculate new basic solution.

9. Go to 1.

The coefficients \( a_{kj} \) and \( c_j \) are those corresponding to the current index set \( J \), not the original one in terms of which the problem is phrased, and are calculated by post-multiplying an original constraint matrix row (or objective row) by \( D^{-1} \).

In the program the heuristic used in step 3 is to pick the most violated constraint. In step 7 a tie for \( k \) is broken randomly, a procedure which prevents cycling almost surely.
TEST PROGRAM

A small test program to run the following examples is listed in Figure 2. The lines between the call to LINOPT and the call to TABLO merely do some cosmetic surgery on the output, replacing quantities near $M$ in magnitude (actually those $\geq \sqrt{M}$) by $+R$ (machine infinities). Three examples are given, with NAMELIST inputs and the outputs from the program. Note that in all three, ROW and COL are defined to correspond to storage by rows, and the columns of the array $A$ contain the rows of the constraint coefficient matrix.

EXAMPLE 1

This example is essentially the problem discussed in Section 1-3 of Hadley\(^7\), with slightly modified coefficients.

Maximize $x_8: = 5.0 \ x_1 + 7.6 \ x_2 + 8.0 \ x_3 + 4.0 \ x_4$

subject to

$x_1 \geq 0, \ x_2 \geq 0, \ x_3 \geq 0, \ x_4 \geq 0,$

$x_5: = 1.5 \ x_1 + 1.2 \ x_2 + 2.4 \ x_3 + 1.2 \ x_4 \leq 2100,$

$x_6: = 1.0 \ x_1 + 4.5 \ x_2 + 1.0 \ x_3 + 3.0 \ x_4 \leq 8000,$

$x_7: = 1.5 \ x_1 + 3.0 \ x_2 + 3.6 \ x_3 + 1.0 \ x_4 \leq 5000.$
NAMELIST INPUT:

\$IN
IOBJ = 8
M = 4,
N = 4,
MIN = .FALSE.,
INVERT = .TRUE.,
ITMAX = 1000,
EPS = 0.,
ROW = 0, 10, 20, 30,
COL = 1, 2, 3, 4,
A(I, 1) = 1.5, 1.2, 2.4, 1.2,
A(I, 2) = 1.0, 4.5, 1.0, 3.0,
A(I, 3) = 1.5, 3.0, 3.6, 1.0,
A(I, 4) = 5.0, 7.6, 8.0, 4.0,
BL = 4*0., 4*-1.E100,
BU = 4*1.E100, 2100., 8000., 5000., 1.E100,
K = 1, 2, 3, 4, 5, 6, 7, 8,
X = 4*0.,
\$END

(Since the objective coefficients $c_j$ are all nonzero for the initial tableau, it is not really necessary to preset $x_1$, $x_2$, $x_3$ and $x_4$, as Example 2 will show).

Output:

<table>
<thead>
<tr>
<th>I</th>
<th>BL(I)</th>
<th>x(I)</th>
<th>BU(I)</th>
<th>T(I, 1)</th>
<th>T(I, 3)</th>
<th>T(I, 5)</th>
<th>T(I, 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>1625.</td>
<td>R</td>
<td>-.125</td>
<td>-.800</td>
<td>-.417</td>
<td>.500</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>R</td>
<td>0.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>125.</td>
<td>R</td>
<td>-1.125</td>
<td>-1.200</td>
<td>1.250</td>
<td>-.500</td>
</tr>
<tr>
<td>5</td>
<td>-R</td>
<td>2100.</td>
<td>2100</td>
<td>0.000</td>
<td>.000</td>
<td>1.000</td>
<td>-.000</td>
</tr>
<tr>
<td>6</td>
<td>-R</td>
<td>7687.</td>
<td>8000</td>
<td>-2.938</td>
<td>-6.200</td>
<td>1.875</td>
<td>.750</td>
</tr>
<tr>
<td>7</td>
<td>-R</td>
<td>5000.</td>
<td>5000</td>
<td>-.000</td>
<td>.000</td>
<td>.000</td>
<td>1.000</td>
</tr>
<tr>
<td>8</td>
<td>-R</td>
<td>1285.</td>
<td></td>
<td>-.450</td>
<td>-2.880</td>
<td>1.833</td>
<td>1.800</td>
</tr>
</tbody>
</table>

The objective variable, $x(8)$, is to be maximized. ITER = 10, IERR = 0.

The tableau gives information about the primary and secondary variables at the final iteration. The primary variables are $x_1$, $x_3$, $x_5$ and $x_7$. The rows of the tableau give the coefficients of the variables expressed in terms of the primary variables. Thus, $x_4 = -1.125 x_1 - 1.2 x_3 + 1.125 x_5 - .5 x_7$.

The dual variables are not printed explicitly but the nonzero ones can be obtained from the objective row: $u_1 = -.450$, $u_3 = -2.880$, $u_5 = 1.833$, $u_7 = 1.800$. (For a minimization problem, these should be negated.)
EXAMPLE 2

This problem also comes from Hadley\textsuperscript{8}. It is his Problem 8-5.

Minimize $x_4 = 3x_1 - 2x_2 + 4x_3$
subject to

$x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$,

$x_5 = 3x_1 + 5x_2 + 4x_3 \geq 7$,

$x_6 = 6x_1 + x_2 + 3x_3 \geq 4$,

$x_7 = 7x_1 - 2x_2 - x_3 \leq 10$,

$x_8 = x_1 - 2x_2 + 5x_3 \geq 3$,

$x_9 = 4x_1 + 7x_2 - 2x_3 \geq 2$.

Input:

$\$IN
IOBJ = 4,
M = 6,
N = 3,
MIN = . TRUE.,
INVERT = . TRUE.,
ITMAX = 1000,
EPS = 0.,
ROW = 0, 10, 20, 30, 40, 50,
COL = 1, 2, 3,
A(l, 1) = 3., -2., 4.,
A(l, 2) = 3., 5., 4.,
A(l, 3) = 6., 1., 3.,
A(l, 4) = 7., -2., -1.,
A(l, 5) = 1., -2., 5.,
A(l, 6) = 4., 7., -2.,
BL = 3*0., -1.E100, 7., 4., -1.E100, 3., 2.,
BU = 6*1.E100, 10., 2*1.E100,
K = 1,2,3,4,5,6,7,8,9,$

$\$END

Output:

TABLEAU

<table>
<thead>
<tr>
<th>I</th>
<th>BL(I)</th>
<th>X(I)</th>
<th>BU(I)</th>
<th>T(I, 1)</th>
<th>T(I, 5)</th>
<th>T(I, 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>R</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>R</td>
<td>-1.667</td>
<td>1.000</td>
<td>-0.152</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>R</td>
<td>-1.667</td>
<td>0.061</td>
<td>-0.152</td>
</tr>
<tr>
<td>4</td>
<td>-R</td>
<td>-R</td>
<td>R</td>
<td>2.333</td>
<td>-0.61</td>
<td>0.848</td>
</tr>
<tr>
<td>5</td>
<td>7.000</td>
<td>R</td>
<td>R</td>
<td>-0.000</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>4.000</td>
<td>R</td>
<td>R</td>
<td>4.667</td>
<td>0.333</td>
<td>0.333</td>
</tr>
<tr>
<td>7</td>
<td>-R</td>
<td>-R</td>
<td>10.000</td>
<td>8.000</td>
<td>-3.64</td>
<td>0.091</td>
</tr>
<tr>
<td>8</td>
<td>3.000</td>
<td>3.000</td>
<td>R</td>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>9</td>
<td>2.000</td>
<td>R</td>
<td>R</td>
<td>2.333</td>
<td>0.939</td>
<td>-1.152</td>
</tr>
</tbody>
</table>

THE OBJECTIVE VARIABLE, $X(4)$, IS TO BE MINIMIZED.

ITER = 3 IERR = 0

\textsuperscript{8} Ibid., p. 267.
This problem has an unbounded solution: \( X_5 = + \infty \) and the objective value is \(-\infty\).

**EXAMPLE 3**

This example illustrates the solution of a dual problem.

Minimize \( \sum_{k=3}^{9} u_k \)

subject to \( \sum_{k=3}^{9} u_k = 1 \),

\( \sum_{k=3}^{9} k u_k = 1 \).

(The indexing starts at 3 for convenience.) Since there is no unit matrix in the constraint coefficient matrix, we add artificial variables \( u_1 \) and \( u_2 \), which must vanish at the solution:

\[
\begin{align*}
    u_1 + \sum_{k=3}^{9} u_k &= 1, \\
    u_2 + \sum_{k=3}^{9} k u_k &= 1.
\end{align*}
\]

Noting that \( |u_k| = \max \{-u_k, u_k\} \), we can transform to the primal problem:

Maximize \( x_{10} = x_1 + x_2 \)

subject to \(-1 \leq x_i \leq 1\), \( i = 3, \ldots, 9 \), where

\[
\begin{align*}
    x_3 &= x_1 + 3x_2, \\
    x_4 &= x_1 + 4x_2, \\
    x_5 &= x_1 + 5x_2, \\
    x_6 &= x_1 + 6x_2, \\
    x_7 &= x_1 + 7x_2, \\
    x_8 &= x_1 + 8x_2, \\
    x_9 &= x_1 + 9x_2.
\end{align*}
\]

The variables \( x_1 \) and \( x_2 \), dual to artificial variables, are not constrained directly.

Input:

\$IN

IOBJ = 10,
M = 8,
N = 2,
MIN = .FALSE.,
INVERT = .TRUE.,
ITMAX = 1000,
EPS = 0.,
ROW = 0, 10, 20, 30, 40, 50, 60, 70,
COL = 1, 2,
A(1, 1) = 1., 3.,
A(1, 2) = 1., 4.,
A(1, 3) = 1., 5.,
A(1, 4) = 1., 6.,
The solution is obtained from row 10: $u_3 = 1.333, u_9 = -0.333, u_4, \ldots, u_8 = 0$. The minimal value is 1.667. (Obviously, the exact solution has $u_3 = 4/3, u_9 = -1/3$.)
<table>
<thead>
<tr>
<th>PROGRAM TEST</th>
<th>OPT = 1</th>
<th>FTN 4.6 + 452</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>PROGRAM TEST (INPUT, OUTPUT)</td>
</tr>
<tr>
<td>2</td>
<td>TEST</td>
<td>LINE</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>TEST PROGRAM FOR LINOPT</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>TEST</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>C</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>INTEGER ROW, COL</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>COMMON /XXLP/</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>INTEGER ROW, COL</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>COMMON /XXLP/</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>INTEGER ROW, COL</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>COMMON /XXLP/</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>INTEGER ROW, COL</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>COMMON /XXLP/</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>INTEGER ROW, COL</td>
</tr>
</tbody>
</table>

NC4NC4C-00~

LOGO

00

H

I

I00

I

C4

26

FIGURE 2 TEST PROGRAM
CHAPTER 6

LISTING

SUBROUTINE LINOPT(A,ROW, COL, BL,BU, N, X, JOBJ, C)

LINE 2
C LINEAR PROGRAMMING BY THE DUAL SIMPLEX ALGORITHM
C
C PROBLEM--
C
C MINIMIZE OR MAXIMIZE X(JOBJ) SUBJECT TO
C
C X(N+I) = SUM(J = 1,...,M) A(ROW(I)+COL(J)) * X(J),
J = 1,...,N,
C
C BL(J) .LE. X(J) .LE. BU(J), J = 1,...,N+M.
C
C FURTHER DOCUMENTATION AND EXAMPLES OF USE CAN BE FOUND IN--
C
C NSWC TR 80-413, LINOPT, A FORTRAN ROUTINE FOR SOLVING
C LINEAR PROGRAMMING PROBLEMS, BY J.W. WINGATE.
C
C ARRAYS ARE PASSED AS FORMAL PARAMETERS, SIMPLE VARIABLES AS
C ELEMENTS OF THE COMMON BLOCK /XAXIS/.
C
C INPUTS--THE FOLLOWING VARIABLES AND ARRAYS MUST BE DEFINED ON
C ENTRY
C
C JOBJ INDEX OF THE OBJECTIVE VARIABLE.  (INTEGER LINOPT 31)
C (NOTE THAT X(JOBJ) IS ALSO CONSIDERED AS
C A CONSTRAINED VARIABLE.)
C
C M NUMBER OF DEPENDENT VARIABLES.  (INTEGER LINOPT 33)
C
C M NUMBER OF INDEPENDENT VARIABLES.  (INTEGER LINOPT 34)
C
C MIN .TRUE. FOR MINIMIZATION,  (LOGICAL LINOPT 35)
C .FALSE. FOR MAXIMIZATION.
C
C 27
INVERTI = .TRUE. IF THE INVERSE MATRIX E IS TO BE CALCULATED.
FALSE. IF E IS ALREADY SET TO THE INVERSE OF THE BASIS DEFINED BY K. (FOR OPTIMIZATION)
INVERT SHOULD BE .FALSE. UNLESS REINVERSION IS DESIRED.)
CONTROL IS RETURNED TO THE CALLING PROGRAM AFTER ITMAX ITERATIONS.
EPS = ZERO TOLERANCE. CONSTRAINT VIOLATIONS ARE TREATED AS ZERO.
A = ARRAY CONTAINING THE COEFFICIENT MATRIX.
ROW INDEX ARRAY.
COL COLUMN INDEX ARRAY.
THE COEFFICIENT OF X(J) IN THE EQUATION FOR X(N+I) IS A(ROW(I)+COL(J)).
Either (CASE 1)
ONE HAS VECTORS ARGW1,...,ARGWN WITH ARGW1(COL(J)) THE COEFFICIENT OF X(J)
IN THE EQUATION FOR X(N+I), IN WHICH CASE ROW(I) = LOCF(ARGW1) - LOCF(A), I = 1,...,M.
OR (CASE 2)
ONE HAS VECTORS ACOL1,...,ACOLN WITH ACOL1(ROW(I)) THE COEFFICIENT OF X(J)
IN THE EQUATION FOR X(N+I), IN WHICH CASE COL(J) = LOCF(ACOLJ) - LOCF(A), J = 1,...,N.
AND THE COEFFICIENT MATRIX IS STORED IN THE FIRST M ROWS AND N COLUMNS OF A (CASE 2),
ROW(I) = I, COL(J) = (J-1)*M, J = 1,...,N,
WHILE IF THE COEFFICIENT MATRIX IS STORED IN THE FIRST N ROWS AND M COLUMNS
(CASE 1),
ROW(I) = (I-1)*M, I = 1,...,M,
COL(J) = J, J = 1,...,N,
ROW AND COL MAY BE PERMUTED IN ANY CONVENIENT WAY.)
BL = ARRAY OF LOWER BOUNDS.
BU = ARRAY OF UPPER BOUNDS.
K = BASIC SOLUTION KEY.
K, IN CONJUNCTION WITH X, SPECIFIES A PARTICULAR BASIC SOLUTION. THE EQUATIONS RELATING X(N+I), I = 1,...,M TO X(J), J = 1,...,N (THE CONSTRAINT EQUATIONS) CAN BE SOLVED FOR VARIOUS COMBINATIONS OF K.
M VARIABLES (SECONDARY VARIABLES) IN TERMS OF THE REMAINING M VARIABLES (PRIMARY VARIABLES). K, A PERMUTATION OF (1,...,N+A), SPECIFIES SUCH A PARTITION INTO PRIMARY AND SECONDARY VARIABLES. K(1),...,K(N) ARE THE INDICES OF THE PRIMARY VARIABLES. K(N+1),...,K(N+M) ARE THE INDICES OF THE SECONDARY VARIABLES. FOR THE DUAL SOLUTION, U(J) AND V(J) ARE THE PRIMARY AND SECONDARY INDICES, RESPECTIVELY. FOR THE DUAL SOLUTION, U(J) MUST BE SET TO EITHER SL(J) OR B(J), J = 1,...,N. THESE ARE DEFAULT VALUES TO BE USED WHEN A VANISHING U(J) MAKES X(J) INDETERMINATE IN SETTING UP A DUAL-FEASIBLE SOLUTION.

PRIMAL SOLUTION ARRAY.

U(J) IS THE DUAL VARIABLE (LAGRANGE MULTIPLIER) FOR THE CONSTRAINTS ON X(J).

IT IS POSITIVE IF THE UPPER BOUND IS ACTIVE.

NEGATIVE IF THE LOWER BOUND IS ACTIVE.

INVERSE MATRIX ARRAY.

X(I) = SUM (J = 1,...,N) E(I,J) * X(J),

I = 1,...,N.
SCR (ALIAS ARROW IN SUBROUTINES) SCRATCH ARRAY.

MINIMUM DECLARED ARRAY SIZES--

A       m * n
ROW     m
COL     n
BL      m * n
DU      m * n
K       m * n
X       m * n
U       m * n
E       n * n
SCR     n

SUBROUTINE TABLO (Q.V.) PRINTS THE FULL EXPLICIT TABLEAU.
IT IS NOT CALLED THROUGH LINOPT AND MUST BE CALLED SEPARATELY.

DIMENSION BL(1), BU(1), K(1), X(1), U(1), SCR(1)

COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR /XXXLP/ 2
COMMON /XXXLP/ HPI, NPM, IPIV, JPIV, NEG
COMMON /XXXLP/ BIGN
LOGICAL MIN, INVERT, NEG

THE VARIABLE BIGN REPRESENTS A VERY LARGE NUMBER. THE DEFAULT
VALUE IS 1.E100. THE USER MAY RESET THIS VALUE IF SO DESIRED.
BIGN OR -BIGN MAY BE USED TO FILL IN MISSING UPPER OR LOWER
BOUNDS.

DATA BIGN /1.E100/

M1 = N + 1
NPM = N * M
IF (.NOT. INVERT) GO TO 10
   CALL SETINV(A, ROW, COL, K, E, SCR)
IF (IERR.EQ.3) RETURN
10 CONTINUE
   CALL GETROJ(A, ROW, COL, E, IOBJ, SCR)
DO 50 J = 1, M
   K(J) = K(J)
IF (MIH) SCR(J) = -SCR(J)
U(KJ) = SCR(J)
IF (U(KJ)) 20, 40, 30
C NEGATIVE
20 CONTINUE
   X(KJ) = BL(KJ)
   GO TO 40
C POSITIVE
30 CONTINUE
   X(KJ) = BU(KJ)
40 CONTINUE
50 CONTINUE
DO 60 J = NPI, NPH
   U(K(J)) = 0.
60 CONTINUE
   CALL PSOL(A,ROW,COL,K,X,E)
   CALL DSIMP(A,ROW,COL,BL,BU,K,X,U,E,SCR)
   IF (IERR.NE.0) GO TO 110
C ROUND X-VALUES WITHIN EPS OF BOUNDS
DO 100 I = 1, NPH
   IF (ABS(X(I)-BL(I)).LE.EPS) X(I) = BL(I)
   IF (ABS(X(I)-BU(I)).LE.EPS) X(I) = BU(I)
100 CONTINUE
110 CONTINUE
RETURN
END
SUBROUTINE DSiTiP(A,ROU,COL,BL,9U,K,X,U,E,AROW)
C
C DUAL SIMPLEX ALGORITHM
C
C DIMENSION BL(1), BU(1), K(1), X(1), U(1), AROW(1)
C
COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
COMMON /XXXLP/ NPI, NPA, IPIV, JPIV, NEGV
COMMON /XXXLP/ @IG?
LOGICAL MIN, INVERT, NEGV
C LINE 2
C ----------------------------------------------------------------------

IERR = 2
C WHEN ITMAX.LT.1 THE LOOP IS PARTIALLY EXECUTED
DO 10 II = 1, ITMAX
   CALL PIVROW(BL,BU,K,X)
   IF (IPIV.GT.0) GO TO 10
   NO PIVOT ROW INDICATES THAT X IS OPTIMAL
   IERR = 0
   RETURN
10 CONTINUE
   KROW = K(IPIV)
   CALL GETROW(A,ROU,COL,E,KROW,AROW)
   CALL PIVCOL(BL,BU,K,X,U,AROU)
   IF (JPIV.GT.0) GO TO 40
   NO PIVOT COLUMN INDICATES THAT THE CONSTRAINTS
   ARE INCONSISTENT
   IERR = 1
   RETURN
40 CONTINUE
   IF (ITMAX.LT.1) RETURN
   NEW SOLUTION KEY
   K(IPIV) = K(JPIV)
   K(JPIV) = KROW
   CALL NEWINV(E,AROW)
   NEW DUAL SOLUTION
   CALL GETROW(A,ROU,COL,E,IOBJ,AROW)
   DO 70 J = 1, M
      IF (MIN) AROW(J) = -AROW(J)
      U(K(J)) = AROW(J)
    70 CONTINUE
   U(K(IPIV)) = 0.
   NEW PRIMAL SOLUTION
   X(KROW) = BU(KROW)
   IF (NEGV) X(KROW) = BL(KROW)
   CALL PSOL(A,ROU,COL,K,X,E)
   ITER = ITER + 1
   CONTINUE
SUBROUTINE PIVROW(L,BU,K,X)

C PIVOT ROW SELECTION

C

C DIMENSION BL(1), BU(1), K(1), X(1)

C COMMON /XXXLP/ IOBJ, N, M, MIN, INVERT, ITMAX, EPS, ITER, IERF

C COMMON /XXXLP/ NPI, NPR, IPIV, JPIV, NEGV

C COMMON /XXXLP/ BIGM

C LOGICAL MIN, INVERT, NEGV

C

C

P = 0
IF (NPI.LT.NP1) RETURN
VII = 0.
DO 50 II = NP1, NPI

I = K(II)

C CHECK CONSTRAINTS ON X(I)

D = X(I) - BL(I)
IF (D.GE.-EPS) GO TO 10
D = -D
IF (VII.GT.D) GO TO 40
VII = D
IPIV = II
NEGV = .TRUE.
GO TO 40

10 CONTINUE
D = X(I) - BU(I)
IF (D.LE.EPS) GO TO 30
IF (VII.GT.D) GO TO 40
VII = D
IPIV = II
NEGV = .FALSE.
30 CONTINUE
40 CONTINUE
50 CONTINUE
RETURN
SUBROUTINE PIVCOL(GL, SU, K, X, U, AROW)

C PIVOT COLUMN SELECTION
C
dimensions BL(1), SU(1), K(1), X(1), U(1), AROW(1)
C common /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
C common /XXXLP/ NP1, NPI, IPIV, JPIV, NEGV
C common /XXXLP/ BIGN
LOGICAL MIN, INVERT, NEGV

JJ = 1, N
J = K(JJ)
   AA = AROW(JJ)
   IF (NEGV) AA = -AA
   IF (AA.GE.0. AND. X(J).EQ.BL(J)) go to 20
   IF (AA.LE.0. AND. X(J).EQ.BU(J)) go to 20
   R = U(J)/AA
   IF (R.GT.W) go to 10
   IF (R.EQ.W .AND. RANF(AA).GT.0.5) go to 10
   JJ = J
   W = R
10 CONTINUE
20 CONTINUE
30 CONTINUE
RETURN
END
SUBROUTINE GETROW(A,ROW,COL,E,KROW,AROW)

GENERATION OF CONSTRAINT COEFFICIENTS FOR THE CURRENT BASIS

DIMENSION A(1), ROW(1), COL(1), E(1), AROW(1)
INTEGER ROW, COL

COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
COMMON /XXXLP/ MPI, NPI, IPIV, JPIV, NEGV
COMMON /XXXLP/ BIGH
LOGICAL MIN, INVERT, NEOV

IF (KROW.GT.N) GO TO 20
ORIGINAL INDEPENDENT VARIABLE. GET ROW KROW OF THE INVERSE.

JJ = 0
DO 10 J = 1, N

AROW(J) = E(KROW+JJ)

IF (ABS(AROW(J)).LE.EPS) AROW(J) = 0.
JJ = JJ + N
10 CONTINUE
GO TO 50

CONTINUE

ORIGINAL DEPENDENT VARIABLE.

MULTIPLY ORIGINAL ROW BY THE INVERSE.

KK = ROW(KROW-N)
JJ = 0
DO 40 J = 1, N

AROW(J) = 0.

DO 30 I = 1, N

AROW(J) = AROW(J) + A(KK+COL(I))*E(I+JJ)
30 CONTINUE

IF (ABS(AROW(J)).LE.EPS) AROW(J) = 0.
JJ = JJ + N
40 CONTINUE
RETURN
END
SUBROUTINE PSOL(A,ROW,COL,K,X,E)

DIMENSION A(1), ROW(1), COL(1), K(1), X(1), E(1)
INTEGER ROU, COL
COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
COMMON /XXXLP/ NPI, NPM, IPIV, JPIV, NEGV
COMMON /XXXLP/ bigm
LOGICAL MIN, INVERT, NEGV

DO 30 I = NP1, NPM
   KI = K(I)
   IF (KI.GT.N) GO TO 20
   X(KI) = 0.
   JJ = 0
   DO 10 J = 1, N
      X(KI) = X(KI) + E(KI+JJ) * X(K(J))
      JJ = JJ + N
   CONTINUE
   10 CONTINUE
30 CONTINUE

DO 60 I = NP1, NPM
   KI = K(I)
   IF (KI.LE.N) GO TO 50
   X(KI) = 0.
   KK = ROW(KI-N)
   DO 40 J = 1, N
      X(KI) = X(KI) + A(KK+COL(J)) * X(J)
   CONTINUE
50 CONTINUE
60 CONTINUE
RETURN
END
SUBROUTINE SETINV(A, ROW, COL, E, KROW, AROW)
C
C INITIAL INVERSE
C
DIMENSION K(1), E(1), AR0W(1)
C
COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
    /XXXLP/ 2
COMMON /XXXLP/ MPI, NPM, IPiV, JPIV, NEGV
    /XXXLP/ 3
COMMON /XXXLP/ BIGN
    /XXXLP/ 4
LOGICAL MIN, INVERT, NEGY
    /XXXLP/ 5
C
SET E TO THE IDENTITY
JJ = 0
DO 20 J = 1, N
    DO 10 I = 1, N
        E(I+JJ) = 0.
        10 CONTINUE
    JJ = JJ + N
    20 CONTINUE
C GENERATE INITIAL INVERSE
DO 30 J = 1, N
    K(J) = -K(J)
    30 CONTINUE
DO 90 JJ = 1, H
    DO 40 J = 1, N
        IF (K(J).LT.0) GO TO 50
        40 CONTINUE
    JJ = JJ + N
    50 CONTINUE
KROW = -K(J)
CALL GETROW(A, ROW, COL, E, KROW, AR0W)
ROWMAX = 0.
DO 70 L = 1, N
    TEST = ABS(AR0W(L))
    IF (K(L).GT.0 .OR. TEST.LT.ROWMAX) GO TO 60
        ROWMAX = TEST
        JPIV = L
    60 CONTINUE
   70 CONTINUE
    IF (ROWMAX.GT.0.) GO TO 30
        IERR = 3
        RETURN
    30 CONTINUE
K(J) = K(JPIV)
K(JPIV) = KROW
CALL NEWINV(E,AROW)
90 CONTINUE
ITER = 0
RETURN
END
SUBROUTINE NEWINV(E,AROW)

C INVERSE UPDATE BY COLUMN OPERATIONS

C DIMENSION E(1), AROW(1)

COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR
COMMON /XXXLP/ NPI, NPH, IPIV, JPIV, NEGV
COMMON /XXXLP/ BIGA
LOGICAL MIN, INVERT, NEGV

JJPIV = (JPIV-1)*N
DO 20 I = 1, N
  EPIV = E(I+JJPIV)/AROW(JPIV)
  JJ = 0
  DO 10 J = 1, N
    E(I+JJ) = E(I+JJ) - EPIV*AROW(J)
    JJ = JJ + N
  10  CONTINUE
  JJPIV = EPIV
  JJ = JJ + N
20  CONTINUE
RETURN
END
SUBROUTINE TABL0(A,ROW,COL,BL,BU,X,E,SCR,KORD)

C TABLEAU PRINTOUT
C KORD IS AN ARRAY OF LENGTH AT LEAST N USED FOR REORDERING
C X(1),...X(N) IN ASCENDING ORDER.
C SEE LINOPT FOR DESCRIPTIONS OF THE OTHER PARAMETERS.
C LINOPT MUST HAVE BEEN CALLED BEFORE CALLING TABL0.

C LINE
C DIMENSION A(I), ROW(I), COL(I), BL(I), BU(I), X(I), E(I)
C DIMENSION SCR(I), KORD(I)
C INTEGER ROW, COL
C COMMON /XXXLP/ IOBJ, M, N, MIN, INVERT, ITMAX, EPS, ITER, IERR /XXXLP/
C COMMON /XXXLP/ NPI, NPM, IPIV, JPIV, NEGV /XXXLP/
C COMMON /XXXLP/ BIGM /XXXLP/
C LOGICAL MIN, INVERT, NEGV

C LINE
C 1 FORMAT (IHI//T55,,TABL0 A B L E A U*ilIX,*,TABL0 I *,5X,*BL(I)*,TABL0 I *
C 2 FORMAT (1HO,I3,1X,3F10.3,1X,10F10.3/(36X,1OF10.3))
C 3 FORMAT (2X,*T(I,*,I3,.I)*)
C 4 FORMAT (///IHO,*THE OBJECTIVE VARIABLE, X(*,I3,*), IS TO BE *,A10)
C LINE
C DO 110 J = 1, N
C KORD(J) = J
C 110 CONTINUE
C DO 130 J = 1, N
C JMIN = J
C DO 120 JJ = J, N
C IF (K(KORD(JJ)).LT.K(KORD(JMIN))) JMIN = JJ
C 120 CONTINUE
C KTEMP = KORD(J)
C KORD(J) = KORD(JMIN)
C KORD(JMIN) = KTEMP
C 130 CONTINUE
C DO 10 J = 1, N
C ENCODE (10,3,SCR(J)) K(KORD(J))
C 10 CONTINUE
C PRINT 1, (SCR(J), J = 1, N)
DO 20 I = 1, NPH
    CALL GETROW(A,ROW,COL,E,I,SCR)
    PRINT 2, I, DL(I), X(I), BU(I), (SCR(KORB(J)), J = 1, M)
20   CONTINUE
    OPT = 10HMAXIMIZED.
    IF (MIN) OPT = 10HMINIMIZED.
    PRINT 4, ICBJ, OPT
    RETURN
END
## DISTRIBUTION

<table>
<thead>
<tr>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**Commanding Officer**

**Naval Air Systems Command**

Attn: Mr. Ralph A'Harrah (Code 5301)

Mr. Dale E. Hutchins (Code 5301C)

Mr. Douglas Kirkpatrick (Code 320D)

Mr. Thomas S. Momiyama (Code ADPO18)

Mr. Richard S. Nisimczyk (Code 5335)

Mr. James Rebel (Code 5314)

Dr. G. A. Heiche (AIR 310A)

**Washington, DC 20361**

**Commanding Officer**

**Naval Air Development Center**

Attn: Mr. Charles R. Abrams (Code 6014)

Mr. S. T. Donley (Code 6014)

Mr. Carmen J. Mazza (Code 6053)

Mr. Edward J. Rickner (Code 6014)

**Warminster, PA 18974**

**Office of Naval Research**

800 N. Quincy Street

Attn: Dr. Stuart L. Brodsky

**Arlington, VA 22217**

**Commander**

**Naval Air Test Center**

Attn: Mr. Anthony Rossetti (Code SA71)

**Patuxent River, MD 20670**

**Commander**

**Naval Weapons Center**

Attn: Dr. R. D. Smith (Code 3911)

**China Lake, CA 93555**

**Professor Anthony Calire**

Drexel University

**Philadelphia, PA 19104**

**Chief of Naval Operations**

Department of the Navy

Attn: Mr. R. Piacesi

**Washington, DC 20350**

43
NSWC TR 80-413

DISTRIBUTION (CONT.)

Defense Technical Information Center
Cameron Station
Alexandria, VA  22314

Copies  12
DATE FILMED 2-8