Experimental Software Package for Linear Programming

1985

Deborah S. Fogal
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

Fogal, Deborah S., "Experimental Software Package for Linear Programming" (1985). Retrospective Theses and Dissertations. 4761.
http://stars.library.ucf.edu/rtd/4761

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
EXPERIMENTAL SOFTWARE PACKAGE
FOR LINEAR PROGRAMMING

BY

DEBORAH S. FOGAL
B.S., Northwest Missouri State University, 1972
M.Ed., Millersville State College, 1981

RESEARCH REPORT
Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1985
ABSTRACT

A software package for linear programming has been developed using the revised simplex and dual simplex algorithms. The design of the program incorporates an experimental change in the dual simplex algorithm. If the entered problem is not primal feasible, a modified dual simplex algorithm is used. The traditional dual simplex method requires an initial dual feasible basis and maintains dual feasibility throughout its application. The experimental change is to ignore this criteria of dual feasibility. The objective then becomes to obtain primal feasibility. Once this is attained, the revised simplex algorithm is applied to obtain optimality, if this has not been reached through use of the dual. This experimental change redirects the goal of the dual simplex method from obtaining objective function optimality to obtaining primal feasibility. Program testing has shown the experimental design to produce correct results for a variety of linear programming problems.

The program is written for an IBM PC using PASCAL for coding. Spreadsheet format and menus provide ease in problem input and output. Devices for output of problem and solution are printer, screen and/or disk. A problem can be saved and retrieved at a later time for editing.
ACKNOWLEDGEMENTS

I wish to thank Dr. Al Pozefsky, Director of the University's STAC office, for his database search of linear programming topics.

I especially want to thank Dr. George Brooks for his patience and guidance throughout this research and my studies at UCF. He has always made learning interesting and a challenge. I have never met a finer teacher than he.
# TABLE OF CONTENTS

**INTRODUCTION** .......................... 1
- Problem Statement ..................... 1
- Project Objective ..................... 2
- Notation ............................. 3

**LINEAR PROGRAMMING BACKGROUND** .......... 5
- Linear Programming Problem ........... 5
- Simplex Method ....................... 9
- Revised Simplex Method ............... 11
- Dual Simplex Method .................. 20

**THE PROGRAM** .......................... 23
- Experimental Change of Dual Simplex Method .......... 23
- Program Heuristics .................... 24
- Program Procedures ................... 25
- Example Problem ..................... 27

**PROGRAM TESTING** ........................ 33
- Introduction ........................ 33
- Problem 1 .......................... 33
- Problem 2 .......................... 35
- Problem 3 .......................... 37
- Problem 4 .......................... 39

**CONCLUSIONS AND RECOMMENDATIONS** ........ 41

**APPENDICES** ............................ 42

**A. COMPARISON OF SIMPLEX, REVISED SIMPLEX AND**
**DUAL SIMPLEX METHODS** ............... 42

**B. USER'S MANUAL** ..................... 46

**C. PROGRAM LISTING** ................... 61

**REFERENCES** .......................... 98
INTRODUCTION

Problem Statement

It has been the apparent inherent nature of man to obtain maximum gain/minimum loss in the conversion of resources to products. Areas such as the military, industry and economics abound with problems whose solution involve optimization techniques. The problem thus becomes how to optimize a given quantity, such as profit or cost of a product, which is constrained by restrictions on the resources necessary to produce the product.

If the quantity to be optimized and resource allocation can be expressed mathematically in terms of linear relations, the problem can be written in linear programming format as follows.

\[
\begin{align*}
\text{maximize/minimize} & \quad z = c'x \\
\text{subject to} & \quad Ax = 0 \\
& \quad x \geq 0
\end{align*}
\]

The purpose of this research is to experiment with the dual simplex method as traditionally applied to linear programming problems.
Project Objective

The objective of this research project is two-fold. The first is to apply software engineering techniques and develop an applicable software package. The package is to be interactive and "user friendly." Coding in Pascal will allow for structured and modular program format.

The second objective encompasses experimentation with linear programming algorithms. In general the program will incorporate the revised simplex and dual simplex methods. A change in the initial conditions for the dual simplex method, suggested by Dr. George Brooks, will be implemented. The project's second objective is to show via application of the software package to linear programming problems that this change does not cause incorrect solutions and perhaps enhances the time to solution in certain cases. The mathematical proof of the validity of the modification of the dual is beyond the scope of this research report.
Notation

scalar
Lower case letter
ex: a

row vector
Lower case letter, underscored, marked with '
ex: a'

column vector
Lower case letter and underscored
ex: a

matrix
Upper case letter
ex: A

LP
Linear programming

LP problem
max/min
subject to
z = c'x
Ax = b
x ≥ 0

m
Number of constraints

n
Number of variables

A
m x n matrix such that
A = a11 a12 ... a1m ... a1n
   a21 a12 ... a2m ... a2n
   .   .   ...   .   ...   .
   .   .   ...   .   ...   .
   am1 am2 ... amm ... amn

B
m x m basis matrix consisting of m linearly independent columns of A
\( B^{-1} \)  \hspace{1cm} \text{Inverse of basis matrix } B

\( \mathbf{x}_B \)  \hspace{1cm} \text{vector containing solution values of basic variables}

\( \mathbf{c}_B \)  \hspace{1cm} \text{vector containing cost coefficients of basic variables}

\( \mathbf{a} \)  \hspace{1cm} \text{m x 1 column vector of } \mathbf{A}

\( r \)  \hspace{1cm} \text{Row number of vector to be removed from basis}

\( k \)  \hspace{1cm} \text{Column number of vector to enter basis}

\( \text{yr} \)  \hspace{1cm} \text{scalar coefficient}
LINEAR PROGRAMMING BACKGROUND

Linear Programming Problem

Linear programming is an optimization method used to find the largest (maximization LP) or smallest (minimization LP) value for a given linear function consisting of an unrestricted number of variables. The values of the variables producing this maximum/minimum value must also satisfy a set of constraints, equalities or inequalities, to which the given linear function is subject. In solving a linear programming problem the objective is to find the absolute maxima/minima if it exists. For this reason, differential calculus methods, which find relative maxima/minima, are not appropriate.

The general linear programming problem is of the following format:

\[
\text{max/min} \quad z = c'x \quad (1)
\]

subject to \quad \begin{align*}
A\mathbf{x} &= \mathbf{b} \quad (2) \\
\mathbf{x} &\geq \mathbf{0} \quad (3)
\end{align*}

In this discussion it will be assumed that the constraints of a linear programming problem are consistent, i.e., rank(A) = m. Also, following are some necessary definitions.

a) Solution \quad A set of values which satisfy the constraints, i.e., (2).
b) Feasible solution
A solution which satisfies the non-negativity requirement (3).

c) Basic solution
Solution to B (m x m matrix formed from m linearly independent columns of A).

d) Basic feasible solution
Solution which satisfies the constraints and non-negativity requirement, (2) and (3), such that the solution contains at most m variables.

e) Optimal solution
Feasible or basic feasible solution that optimizes (1) and produces a finite maxima/minima. Unbounded solutions will not be considered optimal.

The method of solving a linear programming problem can be shown by means of a graphical example and its geometrical interpretation. Consider the following problem.

\[
\begin{align*}
\text{max} & \quad z = 45x_1 + 80x_2 \\
\text{subject to} & \quad x_1 + 4x_2 \leq 80 \\
& \quad 2x_1 + 3x_2 \leq 90 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]
Figure 1. Graphical Example
The shaded area of Figure 1 is the feasible solution space for the problem. This is the space that contains values which will satisfy the constraints and non-negativity requirement. Assigning different values to \( z \) will generate a set of parallel lines, designated as dotted lines on the graph. The objective of linear programming is to find the largest/smallest \( z \) value with at least one point in common with the shaded area. The feasible solution area is a convex region whose boundaries contain extreme points. One of these extreme points, \((x_1, x_2) = (24,14)\), produces the optimal maximum solution for this problem. This is the case with all linear programming problems. If an optimal solution exists, then at least one extreme point will be included in the solution.

The simplex method, used to solve linear programming problems for optimal solutions, was created by George B. Dantzig in 1947. As explained by G. Hadley, this method "is an algebraic iterative procedure which will solve exactly (it is not an approximation method) any linear programming problem in a finite number of steps, or give an indication that there is an unbounded solution" (Hadley 1962). The logic of the simplex method is to move from one extreme point to an adjacent one until a solution is found, if it exists. The selection of each extreme point is such that it will cause the greatest increase/decrease of the objective function in moving from the current extreme point. As the number of extreme points is finite, and \( z \) is a monotonic non-
decreasing/increasing function, the simplex method will eventually come to a conclusion.

**Simplex Method**

The revised and dual simplex methods are used in the program. These methods are based on the simplex method, therefore a brief description of it will be given. This algorithm contains three basic processes: determine an initial basic feasible solution, determine a better basic feasible solution, and test the solution for optimality.

Finding an initial basis can be immediate or require manipulation of the constraints. If the problem is such that all constraints are less than or equal inequalities, the addition of slack variables will create an identity submatrix. This submatrix becomes the initial basis whose inverse is also the identity matrix. Most problems however, will not fall into this category. It is also unlikely that an obvious basis will exist in the A matrix. Searching for m linearly independent vectors in A is not a path normally taken. The approach taken is to manipulate A so as to create an identity matrix basis (Cooper 1974). This requires that A and variables must be augmented such that $Ax = b$ becomes $Ax + Ix_a = b$. The variables $x_{ai}$ are known as artificial variables. The Charnes method and two-phase method produce an initial basis using artificial variables.
If a solution is not optimal, the next step is to replace a current basis vector with a non-basis vector such that the new basis creates a better basic feasible solution. The basis vector to be removed is selected so as to maintain feasibility. The non-basis vector which enters the basis is selected so there is improvement to the objective function. If $z_j$, a scalar value, is defined as (Cooper 1974)

$$z_j = A'B Y_j$$

then the vector to enter the basis can be determined as follows.

**Maximization**

Enter vector $k$:

$$z_k - c_k = \min \{z_j - c_j \mid z_j - c_j < 0\}$$

**Minimization**

Enter vector $k$:

$$z_k - c_k = \max \{z_j - c_j \mid z_j - c_j > 0\}$$

The criterion for selecting the vector leaving the basis is

**Leave vector $r$**:

$$\frac{x_{Br}}{y_{ik}} = \min \left\{ \frac{x_{Bi}}{y_{ik}} \mid y_{ik} > 0 \right\}$$

Optimality is based on whether the objective function can be increased/decreased by a change in basis. Given that
z = c'x it can be shown that the new objective function, z can be written as (Cooper 1974)

\[ z = z + \frac{x_{Br}}{Y_{rk}} \cdot (c_k - z_k) \]

Since feasibility has been maintained, \( x_{Bi} \geq 0 \) for all \( i \), and selection of the leaving vector is based on \( y_{ij} > 0 \), then

\[ \frac{x_{Br}}{Y_{rk}} > 0 \]

The status of \( z_j - c_j \) for all \( j \) non-basic variables becomes the quantity to test for optimality, whether maximization or minimization. For maximization problems, to increase the objective function, \( z_k - c_k \) must be less than zero to insure \( z > z \). The objective function can therefore no longer be optimized when \( z_j - c_j \geq 0 \). For minimization, \( z_k - c_k \) must be greater than zero to insure a decrease, thus \( z_j - c_j \leq 0 \) infers the problem can no longer be optimized.

**Revised Simplex Method**

**Introduction**

In the simplex method a transformation is performed during each iteration for \( y_j \), \( x_B \), \( z_j - c_j \) and \( z \). The majority of the computation is involved with determining \( y_j \), \( j = 1 \ldots n \). Attributes of the revised simplex method are as follows (Bazaraa 1977).

1. Uses the same steps as the simplex method.
2. Requires a \((m+1) \times (m+2)\) array versus \((m+1) \times (n+1)\) for simplex. Since \(n > m\), storage will be saved; this will be significant if \(n\) is much larger than \(m\).

3. Only \(Y_k\) is determined whereas \(Y_j, j = 1 \ldots n\) is calculated at each iteration of the simplex method.

4. The amount of round-off error is reduced by using the original values to calculate \(z_j - c_j\) and \(Y_k\).

A fundamental property of the revised simplex method is the storage of the current basis inverse from which all values of interest can be calculated (Cooper 1974):

\[
Y_k = B^{-1} \ast a_k \\
X_B = B^{-1} \ast b \\
z_j - c_j = c'_B \ast B^{-1} \ast a_j - c_j \\
z = c'_B \ast X_B
\]

**Inverse Matrix**

The revised simplex algorithm is dependent on the knowledge of the current inverse matrix. There are two methods to store the inverse, explicit inverse or product form of the inverse (Murty, 1983). If \(B\) is the basis, the explicit form of inverse stores the inverse tableau as:

\[
B^{-1} = \begin{bmatrix}
1 & -c_B B^{-1} \\
0 & B^{-1}
\end{bmatrix}
\]
The product form of the inverse does not store the inverse tableau but rather the individual pivot matrices which are calculated at each iteration (Murty 1983). The program developed for this project uses the explicit form of inverse.

Once the vector to be removed and the vector to enter the basis are determined, the inverse and $x_B$ can be calculated as follows:

Let $B = (d_1, d_2, \ldots d_r \ldots d_m)$ be a non-singular basis matrix whose inverse is known. Let $d_r$ be the vector to be removed and $a_k$ be the vector to enter with the result being $\hat{B} = (d_1, d_2, \ldots a_k, d_{r+1} \ldots d_m)$. Since $B$ is a basis, $a_k$ can be written as a linear combination of the components of $B$:

$$a_k = y_{1k}d_1 + y_{2k}d_2 + \ldots y_{rk}d_r + \ldots y_{mk}d_m$$

Solving for $d_r$ ($y_{rk} \neq 0$)

$$d_r = -\frac{y_{1k}}{y_{rk}} * d_1 - \frac{y_{2k}}{y_{rk}} * d_2 \ldots \frac{1}{y_{rk}} * a_k \ldots - \frac{y_{mk}}{y_{rk}} * d_m$$
Letting \( \mathbf{s}_k = \begin{bmatrix} -y_{1k} & -y_{2k} & \cdots & 1 & \cdots & -y_{mk} \\ y_{rk} & y_{rk} & \cdots & y_{rk} & \cdots & y_{rk} \end{bmatrix} \) then (1) can be simplified to:

\[
\hat{d}_r = B \ast \mathbf{s}_k \quad (2)
\]

If \( \mathbf{e}_i \) represents a unit vector and \( E \) is an identity matrix whose \( r \)th column is replaced by \( \mathbf{s}_k \), then \( E = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{r-1}, \mathbf{s}_k, \mathbf{e}_{r+1}, \ldots, \mathbf{e}_m) \). Using \( E \), (2) (which represents \( \hat{A} \) columns of \( B \)) becomes: \( \hat{B} = B \ast E \). This then implies

\[
\hat{B}^{-1} = E \ast B^{-1}
\]

Thus the current inverse matrix can be calculated by multiplying the previous inverse by \( E \).
Transformation formulas for $B^{-1}$ and $x_B$ can be derived if a new vector is defined (Cooper 1974). Letting $n_k = s_k - e_r$ then

$$n_k = \begin{cases} -\frac{y_{ik}}{y_{rk}} & i \neq r \\ \frac{1}{y_{rk}} - 1 & i = r \end{cases}$$

Also, if $d_i$ is a column of the known inverse $B^{-1}$, then the columns of $B^{-1}$ can be updated using $\hat{d}_i = d_i + d_{kj}n_k$ such that $r$ indicates removed vector row and $k$ is enter vector column. The vector of solution values $\hat{x}_B$ can be updated via $\hat{x}_B = x_B + x_{Br}n_k$. 
Revised Simplex Steps

The revised simplex algorithm without artificial variables is used in the program. Problems are arranged in a manner that guarantees no artificial variables are needed (see page 24 for explanation). The revised simplex method deviates from the simplex method by considering the objective function as a constraint. Therefore the linear programming problem:

\[
\begin{align*}
\text{max/min} & \quad z = c'x \\
\text{subject to} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

is transformed to (letting \(x_0 = z\)):

\[
\begin{align*}
1x_0 - c_1x_1 - c_2x_2 - \ldots - c_nx_n & = 0 \\
0 + a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & = b_1 \\
0 + a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & = b_2 \\
\vdots & \\
0 + a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n & = b_m
\end{align*}
\]

Linear programming problems normally do not begin with constraints of all equality, i.e., \(Ax = b\), however it is necessary that all be equations when implementing the algorithm. Therefore, prior to constructing the revised
simplex format, all constraints are converted to equations with the use of surplus and slack variables. Values of b are also made positive.

This conversion of the constraints results in the expansion of A to include an identity submatrix. From this an initial basis is obtained which is an identity submatrix I. Because the revised simplex method has incorporated the objective function as a constraint, the basis matrix B is augmented as follows:

\[ B^* = \begin{bmatrix} 1 & -\zeta_B \\ \emptyset & I \end{bmatrix} \]

Therefore

\[ B^{*-1} = \begin{bmatrix} 1 & \zeta_B \\ \emptyset & I \end{bmatrix} \]

This inclusion of the objective function as a constraint results in \( x_B \) and \( y_j \) also being augmented (Cooper 1974).

\[ x_B^* = \begin{bmatrix} z \\ x_B \end{bmatrix}, \quad y_j^* = \begin{bmatrix} z_j - c_j \\ y_j \end{bmatrix} \]

For this paper, \( B^{-1}, x_B \) and \( y_j \) will be used to represent the augmented \( B^{*-1}, x_B^* \) and \( y_j^* \) respectively.

Once the original LP problem has been converted to the revised simplex format, the procedure of finding an optimal solution is as follows.
STEP 1

Calculate $z_j - c_j$ for all vectors not in the basis using the following: $z_j - c_j = \text{row}(l)B^{-1} \cdot a_j$. If all values of $z_j - c_j$ are greater than or equal to zero for maximization, less than or equal to zero for minimization, then a solution has been found. If not, proceed to the next step.

STEP 2

Determine the vector to enter the basis.

Maximization LP:

enter vector $k$:

$z_k - c_k = \min \{z_j - c_j \mid z_j - c_j < 0\}$

Minimization LP:

enter vector $k$:

$z_k - c_k = \max \{z_j - c_j \mid z_j - c_j > 0\}$

STEP 3

Calculate $Y_k$ using the formula

$Y_k = B^{-1} \cdot a_k$

STEP 4

Determine if a solution exists. If $Y_{ik} \leq 0$ for all $i$ the problem is unbounded therefore there is no solution. If there exists at least one $Y_{ik} > 0$, proceed to next step.
STEP 5

Determine the vector to be removed from the basis.

Max/min LP:

remove vector r:

\[ \bar{x}_{Br} = \min \left\{ \frac{x_{Br}}{y_{rk}} \mid y_{ik} > 0 \right\} \]

STEP 6

Transform the inverse and \( x_B \).

a) As \( y_k \) has been calculated, determine \( y_{rk} \).

b) Calculate \( n_k \) such that

\[ n_k = \begin{cases} \frac{-y_{ik}}{y_{rk}} & i \neq r \\ 1 & i = r \end{cases} \]

c) Update \( B^{-1} \)

If \( d_i \) represents a column in \( B^{-1} \) then

\[ \hat{d}_i = d_i + d_{rj}n_k \]

d) Update \( x_B \)

\[ \hat{x}_B = x_B + x_{Br}n_k \]

PROCEED TO STEP 1
**Dual Simplex Method**

**Introduction**

The revised simplex algorithm presented is a primal algorithm. It begins with an initial basis which is primal feasible, i.e., $x_B \geq \emptyset$. Execution of the algorithm proceeds in a manner that preserves primal feasibility while moving the objective function to optimality. An optimal solution is obtained once $z_j - c_j \geq 0$ for maximization, $z_j - c_j \leq 0$ for minimization.

Another method of solving a linear programming problem is the dual simplex algorithm. It is a method of solving the original primal problem using a different set of criteria for implementation and solution although it can incorporate the inverse tableau as in the revised simplex method.

The optimality criteria for a primal problem is $z_j - c_j$. Since

$$z_j - c_j = [ z - c' ] * a_j$$

it can be seen that an optimal solution depends not on the requirement vector, $b$, but on the price vector $c'$ and $a_j$. The dual simplex is based on this logic (Hadley 1962). The initial conditions for the algorithm are that the problem is dual feasible, i.e., $z_j - c_j \geq 0$ | $z_j - c_j \leq 0$ and an initial basic solution exists. The criterion of primal feasibility is relaxed. The algorithm then proceeds to maintain dual
feasibility and obtain primal feasibility and thus optimization.

Initial Basic Solution

"The dual simplex algorithm has never been used as a 'general-purpose' linear programming algorithm because of the apparent difficulty attendant upon finding an initial basic solution with $z_j - c_j \geq 0$ for all $j$. However, neither has much effort been expended in investigating how to do so" (Cooper 1974). The principle experimental feature of the project's program is the elimination of the need for an initial basic solution with $z_j - c_j \geq 0 | z_j - c_j \leq 0$. For this reason, explanation of how to attain this will not be given. However, as this section deals with the "normal" dual simplex method, it is assumed in the steps to follow that an initial basis has been found with $z_j - c_j \geq 0 | z_j - c_j \leq 0$.

Dual Simplex Steps

For the program, the dual simplex algorithm is applied to a problem that has been converted to the revised simplex format and uses the explicit form of the inverse. Once the problem is in the correct format and an initial dual feasible basis has been obtained, the procedure for finding an optimal solution is as follows:
STEP 1

Determine the vector to be removed from the basis.
Max/min LP:
remove vector r :
\[ x_{Br} = \min \{ x_{Bi}, i = 1..m \} \]

STEP 2

Determine if a solution exists. Calculate \( y_{rj} \) for all \( j \) such that:
\[ y_{rj} = \text{Row}(r)B^{-1} * a_j \]

If \( y_{rj} \geq 0 \) for all \( j \), there is no solution. If a solution exists, proceed to next step.

STEP 3

Determine the vector to enter the basis.
Maximization LP:
enter vector \( k \) :
\[ z_k - c_k = \max \left\{ \frac{z_j - c_j}{y_{rj}} \mid y_{rj} < 0 \right\} \]
Minimization LP:
enter vector \( k \) :
\[ z_k - c_k = \min \left\{ \frac{z_j - c_j}{y_{rj}} \mid y_{rj} < 0 \right\} \]

STEP 4

Transform the inverse and \( x_B \) as in the revised simplex method. If \( x_{Bi} \geq 0 \) for all \( i \) then the process terminates, otherwise proceed to step 1.

For a comparison of the simplex, revised simplex and dual simplex methods see Appendix A.
THE PROGRAM

Experimental Change of Dual Simplex Methods

The traditional application of the dual simplex method begins with the determination of an initial dual feasible basis. This in itself can be time-consuming. Methods such as artificial constraints, surplus variable-negative cost, Lemke's method and Dantzig's method can be used (Cooper 1974). For the project, the requirement of a dual feasible basis was experimentally relaxed. That is, an initial basis is determined at the beginning involving the identity matrix, and of course a basis is maintained throughout the algorithm. However, the criterion of dual feasibility, \( z_j - c_j \geq 0 \mid z_j - c_j \leq 0 \), is ignored. The objective of the dual simplex thus becomes to obtain primal feasibility and not necessarily objective function optimality. The changed dual will tolerate a non-improvement or degradation in \( z \). At times the experimental change will attain primal feasibility and optimality, but this is not guaranteed. Once the problem is primal feasible, the revised simplex is used if the LP problem is not yet optimal.
Program Heuristics

The program uses both the revised simplex and dual simplex methods. The algorithms can only be applied to a problem after conversion to constraints of equality. It is known that if all constraints have a relation of less than or equal, the inequalities can be converted to equations via the addition of slack variables and an initial basis matrix is guaranteed. To insure this initial basis, constraints are converted as follows:

1. Convert all original problem constraint equations to two inequalities, one less than or equal, the other greater than or equal.
2. Convert all greater than or equal constraints to less than or equal by multiplication of negative one.
3. Convert the constraints, now all less than or equal, to equalities by adding slack variables.

Once this has been accomplished, the objective function is added to the constraints and the problem is now in revised simplex format. The initial basis is then easily found as it contains the z coefficient, 1, and slack variable coefficients and is therefore the identity matrix.

The second major program decision point is to determine if the problem, now converted, is primal feasible, i.e.,
If this is true, the revised simplex algorithm is applied.

The program will loop through procedures to determine enter vector, remove vector, and updates of inverse and solution vector. At the beginning of each loop, optimality is tested, i.e., $z_j - c_j \geq 0 \ | \ z_j - c_j \leq 0$. If this proves true, the program exits the loop and the solution is ready for output.

If the problem is not primal feasible, the dual simplex method with change is applied. In this process the traditional criteria of dual feasibility is ignored. The program will determine the remove vector then the enter vector, and finally will apply the transformation formulae. At the next iteration, the problem is again tested for primal feasibility. If this proves true, then $z_j - c_j$ are tested for optimality. If optimality has not been attained, the program proceeds with the revised simplex algorithm as the problem is now primal feasible. If the problem is not primal feasible, the dual simplex with change is again applied.

**Program Procedures**

The program is based on three sets of procedures. These sets provide problem input, solution and output.

Input is via spreadsheet format. The program supports full screen movement with the use of the arrow keys. The problem can be viewed as being divided into pages, each page

$x_B \geq 0$. If this is true, the revised simplex algorithm is applied.
allowing for five variables and fourteen constraints. The user can then change pages in an increasing or decreasing direction to enter and edit the problem. Function keys are used to signify page change and end of input.

A second set of procedures solve the problem. In general there is a procedure for each step in the algorithms. Three computational procedures for matrix by matrix multiplication, scalar by matrix multiplication, and addition of matrices are included. These are used in the calculations of $z_j - c_j$, $y_k$, $y_{rj}$, $B^{-1}$ and $x_B$ transformations.

The final set of procedures implement solution output. Output is allowed to three different devices - printer, screen and/or disk.

An example problem showing the heuristics of the program will now be presented. Decision of whether to use the revised simplex or dual simplex with change is based on the feasibility of the problem. If the state of the problem is such that it is primal feasible, $x_B \geq \emptyset$, the revised simplex algorithm is applied; if not primal feasible, the dual simplex with change is used.
Example Problem

minimize \quad z = 4x_1 + 2x_2 + 3x_3 - x_4

subject to \quad x_1 + x_2 + x_3 + x_4 \leq 40
\quad x_1 + x_3 - x_4 \geq 10
\quad x_2 + x_4 \geq 2
\quad x_i \geq 0 \quad i = 1, 2, 3, 4

Initial setup and conditions

Convert constraints to less than or equal by multiplying greater than inequalities by negative one:

\quad x_1 + x_2 + x_3 + x_4 \leq 40
\quad -x_1 - x_3 + x_4 \leq -10
\quad -x_2 - x_4 \leq -2

Add slack variables to convert to equalities:

\quad x_1 + x_2 + x_3 + x_4 + x_5 = 40
\quad -x_1 - x_3 + x_4 + x_6 = -10
\quad -x_2 - x_4 + x_7 = -2

Add objective function to constraints

note: let \( x_0 = z \)

\quad x_0 - 4x_1 - 2x_2 - 3x_3 - x_4 = 0
\quad x_1 + x_2 + x_3 + x_4 + x_5 = 40
\quad -x_1 - x_3 + x_4 + x_6 = -10
\quad -x_2 - x_4 + x_7 = -2
The problem is now in the proper format. Initial conditions are:

basis variables = \( (x_0, x_5, x_6, x_7) \)

nonbasis variables = \( (x_1, x_2, x_3, x_4) \)

\[
A = \begin{bmatrix}
1 & -4 & -2 & -3 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & -1 & 0 & -1 & 1 & 0 & 1 & 0 \\
0 & 0 & -1 & 0 & -1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
b = \begin{bmatrix}
0 \\
40 \\
-10 \\
-2 \\
\end{bmatrix}
\]

\[
B^{-1} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
xB = \begin{bmatrix}
0 \\
40 \\
-10 \\
-2 \\
\end{bmatrix}
\]

\[
\bar{a}_1 = [-4, 1, -1, 0] \\
\bar{a}_3 = [-3, 1, -1, 0] \\
\bar{a}_5 = [0, 1, 0, 0] \\
\bar{a}_7 = [0, 0, 0, 1] \\
\]

\[
\bar{a}_2 = [-2, 1, 0, -1] \\
\bar{a}_4 = [1, 1, 1, -1] \\
\bar{a}_6 = [0, 0, 1, 0] \\
\]

Since \( x_{B2} = -10 \) and \( x_{B3} = -2 \), the problem is not primal feasible, therefore, the dual simplex with change is used.

Calculate \( z_j - c_j \) \( j = 1, 2, 3, 4 \)

\[
z_1 - c_1 = -4 \\
z_3 - c_3 = -3 \\
\]

\[
z_j - c_j = \text{Row (1) } B^{-1} \ast \bar{a}_j \\
z_2 - c_2 = -2 \\
z_4 - c_4 = 1 \\
\]
Note that not all \( z_j - c_j \leq 0 \) therefore the problem is not dual feasible.

**Iteration 1**

a) Determine the vector to leave the basis
\[
\min \{x_{Bi}\} \quad \min \{40, -10, -2\} = -10
\]
x_6 leaves \( r = 2 \)

b) Determine \( Y_{rj} = \text{Row}(r) \ B^{-1} \ * \ a_j \ j = 1, 2, 3, 4 \)
\( Y_{21} = -1 \quad Y_{22} = 0 \quad Y_{23} = -1 \quad Y_{24} = 1 \)

c) Determine the vector to enter the basis
\[
\min \left\{ \frac{z_j - c_j}{Y_{rj}} \mid Y_{rj} < 0 \right\}
\]
\( Y_{rj} < 0 \implies j = 1, 3 \)
\( z_1 - c_1 = -4 \quad Y_{21} = -1 \)
\( z_3 - c_3 = -3 \quad Y_{23} = -1 \)
\[
\min \{4, 3\} = 3
\]
x_3 enters basis \( k = 3 \)

d) Apply transformation formulae

i) \( Y_k = B^{-1} * a_j \)
\[
Y_3 = \begin{bmatrix}
-3 \\
1 \\
-1 \\
0
\end{bmatrix}
\]
\( Y_{rk} = Y_{23} = -1 \)
ii) \[ d_k = \frac{-y_{ik}}{y_{rk}} \quad i \neq r \]
\[ d_k = \frac{1}{y_{rk}} - 1 \quad i = r \]
\[ n_3 = [-3, 1, -2, 0] \]

iii) Update \( B^{-1} \)
\[ \hat{d}_i = d_i + d_{rj} n_k \]
\[ B^{-1} = \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

iv) Update \( \hat{X}_B \)
\[ \hat{X}_B = X_B + X_{Br} n_k \]
\[ \hat{X}_B = \begin{bmatrix} 30 \\ 30 \\ 10 \\ -2 \end{bmatrix} \]

d) Problem conditions are now:
- Basis variables = \((x_0, x_5, x_3, x_7)\)
- Nonbasis variables = \((x_1, x_2, x_6, x_4)\)
- Since \( x_{B4} = -2 \) the problem is not primal feasible, the dual simplex with change will be used.

Calculate \( z_j - c_j \quad j = 1, 2, 6, 4 \)
\[ z_1 - c_1 = -1 \quad z_2 - c_2 = -2 \]
\[ z_6 - c_6 = -3 \quad z_4 - c_4 = -2 \]

All \( z_j - c_j \leq 0 \), therefore the problem is now dual feasible.
Iteration 2

a) Determine the vector to leave the basis
\[
\min \{30, 10, -2\} = -2
\]
x7 leaves \( r = 3 \)

b) Determine \( Y_{rj} \)
\[
j = 1, 2, 6, 4
\]
\[
Y_{31} = 0 \quad Y_{32} = -1 \quad Y_{36} = 0 \quad Y_{34} = -1
\]

c) Determine the vector to enter the basis
\[
\min \{2, 2\} = 2
\]
x2 enters \( k = 2 \)

d) Apply transformation formulae

i) \( Y_2 = \begin{bmatrix} -2 \\ 1 \\ 0 \\ -1 \end{bmatrix} \)
\[
Y_{rk} = Y_{32} = -1
\]

ii) \( n_2 = [-2, 1, 0, -2] \)

iii) Update \( B^{-1} \)
\[
B^{-1} = \begin{bmatrix} 1 & 0 & -3 & -2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}
\]

iv) Update \( \hat{X}_B \)
\[
\hat{X}_B = \begin{bmatrix} 34 \\ 28 \\ 10 \\ 2 \end{bmatrix}
\]

e) Problem conditions are now:

Basis variables = \( (x_9, x_5, x_3, x_2) \)
Non basis variables = \((x_1, x_7, x_6, x_4)\)

Since all \(x_B \geq 0\), the problem is primal feasible.

Calculate \(z_j - c_j\) \(j = 1, 7, 6, 4\)

\[
\begin{align*}
z_1 - c_1 &= -1 \\
z_7 - c_7 &= -2 \\
z_6 - c_6 &= -3 \\
z_4 - c_4 &= 0
\end{align*}
\]

The problem is primal feasible and all \(z_j - c_j \leq 0\) therefore the solution is optimal.

Optimal Solution

Basis variables = \((x_0, x_5, x_3, x_2)\)

\[
\begin{align*}
z &= 34 \\
x_5 &= 28 \\
x_3 &= 10 \\
x_2 &= 2
\end{align*}
\]
PROGRAM TESTING

Introduction

Four problems are presented as examples of program testing. Given below are the problems, initial feasibility status and solutions. The first three problems and solutions were taken from *Methods and Applications of Linear Programming* (Cooper 1974). The fourth problem is the example problem previously given. Following each problem and correct solution is the output of solution determined by the program. Note that feasibility is based on changing all constraints to \( \leq \) then to equations.

**Problem 1**

Problem and Correct Solution

\[
\begin{align*}
\text{max } \quad & z = x_1 + 2x_2 + 3x_3 + 3x_4 \\
\text{s.t.} \quad & 2x_1 + x_2 + 3x_3 + 5x_4 \leq 30 \\
& x_1 + 2x_2 + 4x_3 + 2x_4 \leq 16 \\
& 3x_1 + 2x_2 + 3x_3 + 4x_4 \leq 24 \\
& x_1 \geq 0 \quad \quad i = 1, 2, 3, 4
\end{align*}
\]

Status: Primal feasible

Solution: \( z = 20 \)

\[
\begin{align*}
x_2 &= 4 \\
x_4 &= 4 \\
x_5 &= 6
\end{align*}
\]
PROGRAM OUTPUT

NAME: PROBLEM1
TYPE: MAXIMIZATION
ITERATIONS: 3

\[ Z = 1.00000x(1) + 2.00000x(2) + 3.00000x(3) + 3.00000x(4) \]

\[ 2.00000x(1) + 1.00000x(2) + 3.00000x(3) + 5.00000x(4) \leq 30.00 \]
\[ 1.00000x(1) + 2.00000x(2) + 4.00000x(3) + 2.00000x(4) \leq 16.00 \]
\[ 3.00000x(1) + 2.00000x(2) + 3.00000x(3) + 4.00000x(4) \leq 24.00 \]

VARIABLES

BASIS: x(5) x(2) x(4)

NONBASIS: x(1) x(3) x(6) x(7)

SOLUTION IS

\[ Z = 20.00000 \]
\[ x(5) = 6.00000 \]
\[ x(2) = 4.00000 \]
\[ x(4) = 4.00000 \]
Problem 2

Problem and Correct Solution

\[ \text{max } z = -2x_1 - 3x_2 - x_3 \]
\[ 2x_1 + x_2 + 2x_3 \geq 3 \]
\[ 3x_1 + 2x_2 + x_3 \geq 4 \]
\[ x_i \geq 0 \quad i = 1, 2, 3 \]

Status: Primal infeasible, dual feasible

Solution: \[ z = -2.75 \]
\[ x_1 = 1.25 \]
\[ x_3 = 0.25 \]
PROGRAM OUTPUT

NAME: PROBLEM2
TYPE: MAXIMIZATION
ITERATIONS: 2

\[ z = -2.00000x(1) + -3.00000x(2) + -1.00000x(3) \]

\[ 2.00000x(1) + 1.00000x(2) + 2.00000x(3) \geq 3.00000 \]
\[ 3.00000x(1) + 2.00000x(2) + 1.00000x(3) \geq 4.00000 \]

VARIABLES

BASIS: \( x(3) \)
\( x(1) \)

NONBASIS: \( n(5) \)
\( x(2) \)
\( x(4) \)

SOLUTION IS

\[ z = -2.75000 \]
\[ x(3) = 0.25000 \]
\[ x(1) = 1.25000 \]
Problem 3

Problem and Correct Solution

\[
\begin{align*}
\text{max} & \quad z = 2x_1 - 3x_2 - 2x_3 \\
\text{s.t.} & \quad x_1 - 2x_2 - 3x_3 = 8 \\
& \quad 2x_2 + x_3 \leq 10 \\
& \quad x_2 - 2x_3 \geq 4 \\
& \quad x_i \geq 0 \quad i = 1, 2, 3
\end{align*}
\]

Status: Primal and dual infeasible

Solution: \( z = 22.4 \)

\[
\begin{align*}
x_1 &= 18.8 \\
x_2 &= 4.8 \\
x_3 &= .4
\end{align*}
\]
PROGRAM OUTPUT

NAME: PROBLEM3
TYPE: MAXIMIZATION

ITERATIONS: 4

\[ Z = 2.00000x(1) - 3.00000x(2) - 2.00000x(3) \]

\[ 1.00000x(1) - 2.00000x(2) - 3.00000x(3) = 8.00000 \]
\[ 0.00000x(1) + 2.00000x(2) + 1.00000x(3) \leq 10.00000 \]
\[ 0.00000x(1) + 1.00000x(2) - 2.00000x(3) \geq 4.00000 \]

VARIABLES

BASIS: \( x(7) \)
\( x(3) \)
\( x(2) \)
\( x(1) \)

NONBASIS: \( x(4) \)
\( x(6) \)
\( x(5) \)

SOLUTION

\[ z = 22.40000 \]
\[ x(7) = 0.00000 \]
\[ x(3) = 0.40000 \]
\[ x(2) = 4.80000 \]
\[ x(1) = 18.80000 \]
Problem 4

Problem and Correct Solution

\[
\begin{align*}
\text{min } & \quad z = 4x_1 + 2x_2 + 3x_3 - x_4 \\
\text{s. t. } & \quad x_1 + x_2 + x_3 + x_4 \leq 40 \\
& \quad x_1 + x_3 - x_4 \geq 10 \\
& \quad x_2 + x_4 \geq 2 \\
& \quad x_i \geq 0 \quad i = 1, 2, 3, 4
\end{align*}
\]

Status: Primal and dual infeasible

Solution: \( z = 34 \)

\[
\begin{align*}
x_5 &= 28 \\
x_3 &= 10 \\
x_2 &= 2
\end{align*}
\]
PROGRAM OUTPUT

NAME: PROBLEM4
TYPE: MINIMIZATION
ITERATIONS: 2

\[ z = 4.00000x(1) + 2.00000x(2) + 3.00000x(3) - 1.00000(4) \]

\[
\begin{align*}
1.0000x(1) + 1.0000x(2) + 1.0000x(3) + 1.0000x(4) & \leq 40.00 \\
1.0000x(1) + 0.0000x(2) + 1.0000x(3) - 1.0000x(4) & \geq 10.00 \\
0.0000x(1) + 1.0000x(2) + 0.0000x(3) + 1.0000x(4) & \geq 2.00 
\end{align*}
\]

VARIABLES

BASIS: \( x(5) \), \( x(3) \), \( x(2) \)
NONBASIS: \( x(1) \), \( x(7) \), \( x(6) \), \( x(4) \)

SOLUTION IS

\[
\begin{align*}
z &= 34.00000 \\
x(5) &= 28.00000 \\
x(3) &= 10.00000 \\
x(2) &= 2.00000 
\end{align*}
\]
CONCLUSIONS AND RECOMMENDATIONS

The primary objective of this research was to test an experimental change in the dual simplex method by implementation of this change into a computer program. This has been accomplished. Problems tested thus far have produced correct results when compared to either hand calculations, book solutions and LP88 (Eastern Software Products 1983) solutions. The testing indicates that a relaxation of the dual simplex algorithm's requirement of dual feasibility is a valid approach. It is recommended there be further research into this experimental change including formal proof for verification.

Assuming that the modification is mathematically valid, a second question arises concerning the benefit of implementing this change. Limited testing indicates the experimental modification may offer a decrease in time to solution. As solution time is related to the size and composition of a problem, this decrease will be significant for large problems requiring artificial variables.

Implementation of the product form of the inverse rather than the explicit form as used in the program may benefit the program. Further research is necessary to determine if this would improve memory storage requirements and/or time to solution.
APPENDIX A

COMPARISON OF SIMPLEX, REVISED SIMPLEX AND DUAL SIMPLEX METHODS
Following are tables comparing the simplex, revised simplex and dual simplex methods. Table 1 contains rules to be used with maximization linear programming problems. Table 2 contains rules for minimization linear programming problems. Explanation of each rule is as follows.

1. **feasible** - Rule to determine if the problem is primal feasible (simplex, revised simplex) or dual feasible (dual simplex).

2. **optimal** - Rule to determine if an optimal basic feasible solution has been found.

3. **no solution** - Rule to determine if an optimal basic feasible solution exists.

4. **enter vector** - Rule to select the vector to enter the basis.

5. **leave vector** - Rule to select the vector to leave the basis.
### TABLE I

**MAXIMIZATION**

<table>
<thead>
<tr>
<th></th>
<th><strong>SIMPLEX</strong></th>
<th><strong>REVISED SIMPLEX</strong></th>
<th><strong>DUAL SIMPLEX</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>feasible</td>
<td>$x_B \geq 0$</td>
<td>$x_B \geq 0$</td>
<td>$z_j - c_j \geq 0$</td>
</tr>
<tr>
<td>optimal</td>
<td>$z_j - c_j \geq 0$</td>
<td>$z_j - c_j \geq 0$</td>
<td>$x_B \geq 0$</td>
</tr>
<tr>
<td>no solution</td>
<td>$Y_{ik} \leq 0$</td>
<td>$Y_{ik} \leq 0$</td>
<td>$Y_{rj} \geq 0$</td>
</tr>
<tr>
<td>enter vector</td>
<td>$\min z_j - c_j$ \text{ such that } $z_j - c_j &lt; 0$</td>
<td>$\min z_j - c_j$ \text{ such that } $z_j - c_j &lt; 0$</td>
<td>$\max z_j - c_j$ \text{ such that } $Y_{rj} &lt; 0$</td>
</tr>
<tr>
<td>leave vector</td>
<td>$\min \frac{x_{Bi}}{Y_{ik}}$ \text{ such that } $Y_{ik} &gt; 0$</td>
<td>$\min \frac{x_{Bi}}{Y_{ik}}$ \text{ such that } $Y_{ik} &gt; 0$</td>
<td>$\min x_{Bi}$ \text{ such that } $x_{Bi} &lt; 0$</td>
</tr>
</tbody>
</table>
### TABLE II

#### MINIMIZATION

<table>
<thead>
<tr>
<th></th>
<th><strong>SIMPLEX</strong></th>
<th><strong>REVISED SIMPLEX</strong></th>
<th><strong>DUAL SIMPLEX</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>feasible</td>
<td>$x_B \geq 0$</td>
<td>$x_B \geq 0$</td>
<td>$x_B \geq 0$</td>
</tr>
<tr>
<td>optimal</td>
<td>$z_j - c_j \leq 0$</td>
<td>$z_j - c_j \leq 0$</td>
<td>$z_j - c_j \leq 0$</td>
</tr>
<tr>
<td>no solution</td>
<td>$Y_{ik} \leq 0$</td>
<td>$Y_{ik} \leq 0$</td>
<td>$Y_{rj} \geq 0$</td>
</tr>
<tr>
<td>enter</td>
<td>$\max z_j - c_j$</td>
<td>$\max z_j - c_j$</td>
<td>$\min z_j - c_j$</td>
</tr>
<tr>
<td>vector</td>
<td>such that $z_j - c_j &gt; 0$</td>
<td>such that $z_j - c_j &gt; 0$</td>
<td>such that $Y_{rj} &lt; 0$</td>
</tr>
<tr>
<td>leave</td>
<td>$\min x_{Bi}$</td>
<td>$\min x_{Bi}$</td>
<td>$\min x_{Bi}$</td>
</tr>
<tr>
<td>vector</td>
<td>$\frac{1}{Y_{ik}}$</td>
<td>$\frac{1}{Y_{ik}}$</td>
<td>such that $x_{Bi} &lt; 0$</td>
</tr>
</tbody>
</table>
APPENDIX B

USER MANUAL
USER MANUAL

Introduction

This manual contains information concerning requirements for using the software package and how to use the package. The package handles two types of problems, designated as new problem and old problem. A new problem is considered one that has not been previously entered and saved. An old problem is one that has been saved. Handling of an old problem is divided into two parts. One is to output the problem and solution as it was saved, the other is to edit the problem then output the new solution.

Explanation of how to use the package has been divided into four processes. They are: input new problem, output new problem, input old problem with editing, and output old problem with no editing. An example problem with solution is given at the end of the user manual. It is suggested the user input the problem the first time the package is used and compare the given results with those obtained.

Hardware Requirements

The package has been written for use on an IBM PC. A minimum of 256K is advised. Only one disk drive is necessary. A printer is not required as the user has the option to select output to printer, screen and/or disk. It is
necessary to have two disks: program disk and disk for saving problems.

Program Limitations

There are three program limits related directly to the linear programming problem to be solved; they are number of constraints, number of variables, and character size of numeric values to be input. The maximum number of variables is 25. The maximum number of constraints is based on the number of equalities in the problem. Two times the number of equalities plus the number of inequalities ( ≤ or ≥ ) must be less than or equal to 50. The maximum character size for a number is 6. Decimal points and/or negative signs are included in this count. For example 521.36 and -.1234 are considered maximum size.

Keys

Input of menu selection, change of spreadsheet page, and signal that problem has been entered is done via the function keys, Fl . . . Fl0. Movement on the screen is accomplished by using the arrow keys on the numeric keypad. Numbers can not be entered using this keypad when it is set for arrow key movement. The delete, backspace and carriage return key can also be used during input of the problem to the spreadsheet.
Program Startup

Explanation of startup is based on an IBM PC system with two drives. Place DOS 2.x in drive A (left drive). Turn on system unit, monitor then printer. Enter date and time. Remove DOS diskette and replace with program disk. Enter RDS and wait for program to load. Once loaded, a menu will appear on the screen. If problems are to be saved on a separate disk, remove the program disk and insert second disk. Check the numeric keypad for status, i.e., when pressing an arrow key a number should not appear on the screen. If a number does appear, press the Num Lock key. It is advisable to use the arrow/number key pad only for movement on the screen. Numbers should be entered using the main keyboard.

Data Required

1. Linear programming problem in original form.
2. Number of structural variables and constraints.
3. Name of problem. It must be eight or less alphabetic characters.
4. Yes or no for right-hand side analysis.
5. Yes or no for intermediate results.
6. Output form of solution.
Input: New problem

The following main menu should be present on the screen.

<table>
<thead>
<tr>
<th>ENTER SELECTION USING FUNCTION KEYS MARKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1  :  INPUT PROBLEM</td>
</tr>
<tr>
<td>F2  :  OUTPUT SOLUTION</td>
</tr>
<tr>
<td>F3  :  EXIT PROGRAM</td>
</tr>
</tbody>
</table>

Press the function key marked F1 to begin problem input. A second menu will appear on the screen.

<table>
<thead>
<tr>
<th>ENTER SELECTION USING FUNCTION KEYS MARKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1  :  NEW PROBLEM</td>
</tr>
<tr>
<td>F2  :  OLD PROBLEM</td>
</tr>
<tr>
<td>F3  :  EXIT</td>
</tr>
</tbody>
</table>

Again press the F1 function key. The user will then be asked to input information about the problem and what is to be done. Screen will show the following lines one at a time.
MAXIMIZATION PROBLEM? Y)es or N)o Y
ENTER NUMBER OF STRUCTURAL VARIABLES 1
ENTER NUMBER OF CONSTRAINTS 1
ENTER PROBLEM NAME
RIGHT HAND SIDE RANGE ANALYSIS Y)es or N)o N
DO YOU WANT INTERMEDIATE RESULTS Y)es or N)o N

Once the intermediate results question has been answered, the next step will be to input the objective function. The screen will show

ENTER OBJECTIVE FUNCTION

\[ z = x_1 \times 2 x_3 x_4 \]

F1 : ACCEPT LINE/GRID  F2 : PAGE HOME  F3 : PAGE RIGHT
F4 : PAGE LEFT  F5 : PAGE DOWN  F6 : PAGE UP
Enter cost coefficients of the objective function using arrow keys for movement. Once all are entered, press the F1 function key. Next the user is to enter the constraints. The screen will appear as follows

Enter all constraint coefficients, relations (<, =, >) and right-hand side values using arrow keys. To enter ≥ use >; for ≤ use <. Once completed, press the F1 function key. At this point the problem has been entered and the program is in the process of solving it. If there is not a solution, the user will be given this information on the screen. If a solution is found, the main menu will appear. The user can then proceed with output.
Output: New Problem

The main menu should be on the screen.

<table>
<thead>
<tr>
<th>ENTER SELECTION USING FUNCTION KEYS MARKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 : INPUT PROBLEM</td>
</tr>
<tr>
<td>F2 : OUTPUT SOLUTION</td>
</tr>
<tr>
<td>F3 : EXIT PROGRAM</td>
</tr>
</tbody>
</table>

Press the F2 function key for output, a second menu will appear.

<table>
<thead>
<tr>
<th>ENTER SELECTION USING FUNCTION KEYS MARKED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 : OUTPUT TO PRINTER</td>
</tr>
<tr>
<td>F2 : OUTPUT TO SCREEN</td>
</tr>
<tr>
<td>F3 : OUTPUT TO DISK</td>
</tr>
<tr>
<td>F4 : EXIT</td>
</tr>
</tbody>
</table>

Select either printer, screen or disk for output device using appropriate function key. To save a problem on disk, select output to disk. Once output has completed, the main menu will again appear. If another form of output is required, press the F2 key once again. The user will then have the option of output devices.
**Input:** Old problem with editing

The main menu should be present on the screen.

```
ENTER SELECTION USING FUNCTION KEYS MARKED

F1 : INPUT PROBLEM
F2 : OUTPUT SOLUTION
F3 : EXIT PROGRAM
```

Press the F1 function key for input. The second menu to appear will be

```
ENTER SELECTION USING FUNCTION KEYS MARKED

F1 : NEW PROBLEM
F2 : OLD PROBLEM
F3 : EXIT
```

Now press the F2 function key, as this is a problem that has been saved, the user will be asked to input the name
of the file under which it was saved. Once this information has been entered, a third menu will appear.

```
ENTER SELECTION USING FUNCTION KEYS MARKED

F1 : EDIT PROBLEM
F2 : OUTPUT SOLUTION
F3 : EXIT
```

To edit the problem, press the F1 function key. The user will be asked to input information on the problem and what is to be done. The screen will show the following lines with the previous values shown.

```
MAXIMIZATION PROBLEM? Y)es or N)o  Y
ENTER NUMBER OF STRUCTURAL VARIABLES 4
ENTER NUMBER OF CONSTRAINTS 3
ENTER PROBLEM NAME
RIGHT HAND SIDE RANGE ANALYSIS Y)es or N)o  N
DO YOU WANT INTERMEDIATE RESULTS Y)es or N)o  N
```
After all the information has been entered, the objective function will appear on the screen.

\[
z = x_1 x_2 x_3 x_4
\]
\[
1.00000 \quad 2.00000 \quad 3.00000 \quad 3.00000
\]

F1: ACCEPT LINE/GRID  F2: PAGE HOME  F3: PAGE RIGHT
F4: PAGE LEFT  F5: PAGE DOWN  F6: PAGE UP

The user can now edit any values. Once completed press F1. The screen will then display the constraints.
Again, values can be edited. If the number of constraints has increased, numbered blank lines will appear where the new constraints are to be added. Once completed, press F1. The problem has now been entered and the main menu will appear for selection of output.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00000</td>
<td>1.00000</td>
<td>3.00000</td>
<td>5.00000</td>
<td>&lt; 30.00000</td>
</tr>
<tr>
<td>2</td>
<td>1.00000</td>
<td>2.00000</td>
<td>4.00000</td>
<td>2.00000</td>
<td>&lt; 16.00000</td>
</tr>
<tr>
<td>3</td>
<td>3.00000</td>
<td>2.00000</td>
<td>3.00000</td>
<td>4.00000</td>
<td>&lt; 24.00000</td>
</tr>
</tbody>
</table>

F1 : ACCEPT LINE/GRID  F2 : PAGE HOME  F3 : PAGE RIGHT
F4 : PAGE LEFT  F5 : PAGE DOWN  F6 : PAGE UP
Output: Old problem, no edit

The main menu should be on the screen.

```
ENTER SELECTION USING FUNCTION KEYS MARKED

F1 : INPUT PROBLEM
F2 : OUTPUT SOLUTION
F3 : EXIT PROGRAM
```

To output the solution of an old, saved, problem it is necessary to input the problem first. Therefore press the F1 key. A second menu will appear.

```
ENTER SELECTION USING FUNCTION KEYS MARKED

F1 : NEW PROBLEM
F2 : OLD PROBLEM
F3 : EXIT
```

Press the F2 key to input the old problem. The user will then be asked to enter the filename under which it was saved. Once entered, a third menu will appear on the screen.
As this process involves no editing, the F2 key should be pressed. Once this is done, the output menu will appear.

The user should then select the desired form of output. Once output has been completed, the main menu will appear.
If another form of output is required, F2 should be pressed. The output menu will then appear and the user can select the output device.

Example Problem

\[
\begin{align*}
\text{min} & \quad z = 4x_1 + 2x_2 + 3x_3 - x_4 \\
\text{s.t.} & \quad x_1 + x_2 + x_3 + x_4 \leq 40 \\
& \quad x_1 + x_3 - x_4 \geq 10 \\
& \quad x_2 + x_4 \geq 2 \\
\end{align*}
\]

Solution:
\[
\begin{align*}
z &= 34 \\
x_2 &= 2 \\
x_3 &= 10 \\
x_5 &= 28
\end{align*}
\]
APPENDIX C

PROGRAM LISTING
PROGRAM
RDS (input, output);

VAR
matrix = array[1..52,1..76] of real;

VAR
num_var, num_cons : integer;
AMATRIX, BINVERSE : matrix;
RHS, XBI, DUMMY, CLEAR_SIN : single;
NONBASIS, BASIS, SIGN, INDEX_TEST : isingle;
RELATIONS : isingle;
RANGE : array[1..4,1..52] of real;

PROCEDURE Init;

var dim_c, dim_v : integer;
i, j : integer;
begin {init}
iteration := 0;
it_limit := 40;
in_info := false;
printer := false;
if old = false then begin

62
minmax := true;
range_yes := false;
intermediate := false;
num_cons := 1;
num_var := 1;
nosoln := false;
end;
name := 'PROBLEM';
print_old := false;
dim_c := 52;
dim_v := 76;
for j := 1 to dim_v do
  for i := 1 to dim_c do
    begin {3}
      BINVERSE[i, j] := 0;
      if old <> true then AMATRIX[i, j] := 0;
    end; {3}
  for j := 1 to dim_c do
    begin {2}
      NONBASIS[j] := 0;
      BASIS[j] := 0;
      XB1[j] := 0;
      CLEAR_SIN[j] := 0;
      RANGE[1, j] := 0;
      RANGE[2, j] := 0;
      RANGE[3, j] := 0;
      RANGE[4, j] := 0;
      if old <> true then begin {3}
        RELATIONS[j] := 0;
        SIGN[j] := 0;
        RHS[j] := 0;
        end {3}
      else SIGN[j] := RELATIONS[j];
    end; {2}
equal := 0;
route := 0;
done := true;
roundoff := 0.00001;
fore := 6;
aft := 5;
decimal := 0;
end; {init}
{***********************************************************}
PROCEDURE
begin {message}
  messages[1] := 'FUNCTION KEYS MARKED';
  messages[2] := 'PROBLEM';
  messages[3] := 'SOLUTION';
  ENTER SELECTION USING
        F1 : INPUT
        F2 : OUTPUT
        F3 : EXIT

PROGRAM
messages[5] := '; supplies
PROBLEM
messages[6] := '; PROBLEM
messages[7] := '; supplies
messages[8] := '; PROBLEM
messages[9] := '; SOLUTION
messages[10] := '; OUTPUT TO PRINTER
messages[11] := '; OUTPUT TO SCREEN
messages[12] := '; OUTPUT TO DISK
messages[13] := '; SOLUTION
messages[15] := 'end; {message}

PROCEDURE color (x1,x2,y1,y2 : integer);
var i,j : integer;
begin {color}
TEXTCOLOR(15);
TEXTBACKGROUND(15);
for i := x1 to x2 do
  for j := y1 to y2 do begin
    gotoxy(i,j);
    writeln(' ');
  end;
TEXTCOLOR(15);
TEXTBACKGROUND(0);
end; {color}

PROCEDURE Grid (all : boolean);
var var_label, const_label : integer;
i : integer;
begin {grid}
for i := 1 to 5 do
  begin {2}
    var_label := 5*pagex - 5 + i;
    gotoxy(i*10,4);
    if var_label <= num_var then write('X',var_label);
  end; {2}
writeln(' ');
if (all = true) then
  begin {3}
    if num_var <= 5*pagex then begin
      gotoxy(58,4);
    end;
  end; {3}
write('<,=,>')
gotoxy(70,4)
write('RHS')
end;
for i := 1 to 14 do
begin {4}
const_label := 14*pagey - 14 + i;
gotoxy(5,i+4);
if const_label <= num_cons then
writeln(const_label);
end; {4}
end; {grid}
PROCEDURE Arrow (inkey : char;
var x_trunc : integer);
var count_digit:integer);
begin {arrow}
count_digit := 0;
decimal := 0;
x_trunc := TRUNC(wherex/10)*10;
case ord(inkey) of
72 : if wherey > 5 then
gotoxy(x_trunc, wherey-1)
else
write(chr(7));
75 : if x_trunc >= 20 then
gotoxy(x_trunc-10, wherey)
else
write(chr(7));
77 : if x_trunc <= 60 then
gotoxy(x_trunc+10, wherey)
else
write(chr(7));
80 : if wherey < 20 then
gotoxy(x_trunc,wherey+1)
else
write(chr(7));
end; {case}
end; {arrow}
PROCEDURE Key_in (key2 : char;
var starts : integer);
var i,j : integer;
PROCEDURE Ouit;
begin
SCREEN[wherex,wherey] := key2;
i := wherex;
j := wherey;
if starts = 1 then write(' ');
gotoxy(i,j);
write(key2);
starts := starts + 1;
end;
begin {key_in}
case ord(key2) of
8: begin
  gotoxy(wherex-1,wherey);
  write(' ');
  gotoxy(wherex-1,wherey);
  if SCREEN[wherex,wherey] = '.' then decimal := 0;
  SCREEN[wherex,wherey] := '*';
  starts := starts - 1;
end;
13: begin
  gotoxy(10,wherey+1);
  starts := 0;
  decimal := 0;
end;
45,46,48..57,60..62 :
case wherex of
  1..59,70..79 : case ord(key2) of
    60..62 : write(chr(7));
    45 : if starts > 0
         then write(chr(7))
         else OUTIT;
    46 : if decimal=0 then begin
         decimal := decimal + 1;
         OUTIT;
         end
         else write(chr(7));
    48..57 : OUTIT;
  end; {case}
  60..69 : begin
    if (ord(key2) in [45,46,48..57]) then
      write(chr(7)) else OUTIT;
  end;
end; {case}
end; {key_in}
{*********************************************************** }
PROCEDURE Convert (graph3:number; var code:real;
  var i,j,k : integer;
  result : integer;
  stl : string[7];
  negative : integer;
begin
if graph3[1] <> '*' then begin
  if graph3[1] = '-' then begin
    j := 2;
    negative := -1;
end;
end
else begin
    j := 1;
    negative := 1;
end;
stl := '0';
for i := j to 6 do
    if graph3[i] <> '*' then
        stl := stl + graph3[i];
    VAL(stl, code, result);
    code := negative * code;
    if result = 0 then changeit := true;
end
else begin
    code := 0;
    changeit := false;
end;
end {convert}

************
PROCEDURE Whiteout (x, y: integer);
begin {whiteout}
go_to_xy(x, y);
    if in_info=true then
        if x > 60 then write(' ')
            else write(' ');
go_to_xy(x, y);
end; {whiteout}

************
PROCEDURE Page_out;
var row, row_end: integer;
    col, i, j: integer;
    change: integer;
    temp: real;
begin {page_out}
decimal := 0;
case obj of
    false: begin
        row := 1;
        row_end := 1;
        change := -1;
    end;
    true: begin
        row := 14*pagey - 12;
        row_end := 14;
        change := 1;
    end;
end; {case}
for i := 1 to row_end do
    begin {1}
        if row <= num_cons+1 then
            begin {2}
                col := 5*pagex - 3; {variable number}
for j := 1 to 5 do
if col <= num_var+1 then
begin {3}
temp := AMATRIX[row,col];
gotoxy(j*10,i+4);
if temp <> 0.0 then
if temp < 1.0 then write(change * temp:fore:aft)
else write(change * temp:fore:4);
col := col + 1;
end; {3}
if (obj = true) and (num_var<=5*pagex) then begin

goxy(60,4+i);
case RELATIONS[row] of
   -1 : write(' < ');
   0 : write(' = ');
   1 : write(' > ');
end; {case}
temp := RHS[row];
goxy(70,4+i);
if temp <> 0.0 then
if temp < 1.0 then write(temp:fore:aft)
else write(temp:fore:4);
end;
row := row + 1;
end; {2}
end; {page_out}

*************************************************************/
PROCEDURE Page_in;
var graph4 : number;
results : real;
var_end,var_position : integer;
const_end,const_position : integer;
alter : boolean;
i,j,k : integer;

begin
decimal := 0;
var_position := 5 * pagex - 4;
const_position := 14 * pagey - 13;
if num_var < 5*pagex
   then var_end := 5-(5*pagex - num_var)
   else var_end := 5;
if num_cons < 14*pagey
   then const_end := 14 - (14*pagey - num_cons)
   else const_end := 14;
if obj = false
   then begin
      AMATRIX[1,1] := 1;
      .SIGN[1] := 0;
      for i := 1 to var_end do
         begin
            for j := 0 to 5 do
   
}
graph4[j+1] := SCREEN[i*10 + j, 5];
CONVERT(graph4, results, alter);
if alter = true
then AMATRIX[1, var_position + i] := -1 * results;
end;
end
else begin
for j := 1 to const_end do begin {LEFT SIDE CONSTRAINTS}
for i := 1 to var_end do begin
for k := 0 to 5 do
  graph4[k+1] := SCREEN[i*10 + k, 4 + j];
CONVERT(graph4, results, alter);
if alter = true then
  AMATRIX[const_position+j, var_position+i] := results;
end;
end; {LEFT}
for i := 1 to const_end do begin {SIGNS}
case SCREEN[60, 4+i] of
  '<' : SIGN[const_position+i] := -1;
  '=' : SIGN[const_position+i] := 0;
  '>' : SIGN[const_position+i] := 1;
end; {case}
RELATIONS[const_position+i] := SIGN[const_position+i];
end; {SIGNS}
for i := 1 to const_end do begin {RHS}
for k := 0 to 5 do
  graph4[k+1] := SCREEN[70+k, 4+i];
CONVERT(graph4, results, alter);
if alter = true then
  RHS[const_position+i] := results;
end; {RHS}
end; {PAGE_IN}

***********
PROCEDURE Matrix_in (xs, xe, ys, ye: integer;
  var continue, over: boolean;
  i, k, j, result: integer;
  keyl: char;
  count, inc: integer;
begin {matrix_in}
for i := 1 to 20 do
  for j := 1 to 80 do
    SCREEN[j, i] := '*';
over := false;
count := 0;
decimal := 0;
continue := true;
gotoxy(10, 5);
repeat
read(kbd, keyl);
if ord(keyl)=27
then begin {l}
read(kbd,keyl);
case ord(keyl) of
59,60,61,62,63,64 : over := true;  {FUNC KEY, READ OVER}
72,75,77,80 : ARROW(keyl,count);  {ARROW KEY}
83 if old=false then begin {DELETE KEY}
decimal := 0;
k := wherex;
j := wherey;
WHITEOUT(k,j);
for i := 0 to 6 do
SCREEN[k+i,wherey] := '*';
end
else begin
if obj=false then inc:= -1 else inc:= 0;
decimal := 0;
k := trunc(wherex/10)*10;
j := wherey;
WHITEOUT(k,j);
k := trunc(wherex/10) + (5*pagex) -4;
j := 14*pagey + wherey - 17;
case wherex of
10..59 : AMATRIX[j+inc,k] := 0;
60..69 : begin
SIGN[j+inc] := 0;
RELATIONS[j+inc] := 0;
end;
70..76 : RHS[j+inc] := 0;
end; {case}
end;
else
writeln(chr(7));  {ILLEGAL KEY}
end; {case}
continue := true;
end {l}
else begin
if count > 6 then
if (ord(keyl) = 8) or (ord(keyl) = 13) then continue := true
else begin
write(chr(7));
continue := false;
end;
if continue = true then
if (ord(keyl) in [8,13,45,46,48..57,60..62])
then KEY_IN(keyl,count)
else write(chr(7));  {UNACCEPTABLE CHARACTER}
end;
until over = true;
count := 0;
ordkey := ord(keyl);
clrscr;
PROCEDURE Error (gridx, gridy : integer;
select : integer;
one : integer;
var calc : integer);

var code : integer;
ok : boolean;
answer : string[3];
convertl : string[3];

begin
ok := false;
if select = 1 then begin
    WHITEOUT(gridx,gridy);
    write(chr(7));
end
else begin
    repeat
        readln(answer);
        VAL(answer,calc,code);
        if ((old=true) and (length(answer)=0)) or
           ((length(answer)>0) and (length(answer)<=2) and
            (calc>0) and (calc <= one) and (code=0))
            then ok := true
            else begin
                WHITEOUT(gridx,gridy);
                write(chr(7));
            end;
    if (old=true) and (length(answer)=0) then calc := 0;
    until ok = true;
end;

PROCEDURE Change_page (F_key : integer);
begin
    case F_key of
        60 : begin
            pagex := 1;
pagey := 1;
decimal := 0;
        end;
        61 : begin
            if (5*pagex < num_var) then begin
                pagex := pagex + 1;
                decimal := 0;
            end
            else
                write(chr(7));
        end;
        62 : if pagex = 1 then write(chr(7)) else begin
            pagex := pagex - 1;
        end;
        63 :
    end;
decimal := 0;
end;
end;
63 : begin
if (14*pagey < num_cons) then begin
pagey := pagey + 1; {F5 page down}
decimal := 0;
end
else
write(chr(7));
end;
end;
{case}
end;
{******************************************************************************************}
PROCEDURE Change;
var answer : string[2];
i,x,y,z : integer;
numbers : string[3];
result,code : integer;
ok : boolean;
{------------------INNER PROCEDURE-------------------------}
PROCEDURE Check;
begin
repeat
readln(answer);
if (length(answer)<>0) then
if (answer[1] in ['Y','y','N','n']) and (length(answer)<2)
then ok := true else ERROR(x,y,1,0,z)
else ok := true;
until ok;
writeln(' ');
end;
begin {l}
clrscr;
anwer := ' ';
ok := false;
gotoxy(10,10);
x := 61;
y := 10;
write(' SET CONSTRAINT(S) TO ALL ZEROES Y/N? ');
CHECK;
if answer[1] in ['Y','y'] then begin {2}
gotoxy(22,12);
writeln('ENTER CONSTRAINT NUMBER THEN RETURN ');
gotoxy(21,14);
writeln('END WITH NO NUMBER AND CARRIAGE RETURN ');
repeat
write(' ');
x := wherex;
y := wherey;
read(numbers);
if length(numbers) <> 0 then begin {3}
  VAL(numbers,result,code);
  if (code=0) and ((result>0) and (result <= num_cons+l)) then
    for i := 1 to num_var + 1 do begin {8}
      AMATRIX[result+1,i] := 0.0;
      SIGN[result+1] := 0;
      RELATIONS[result+1] := 0;
      RHS[result+1] := 0;
    end {8}
  else ERROR(x,y,1,0,z);
end; {3}
writeln(' ');
until length(numbers) = 0;
end; {2}
clearscr;
answer := ' ';
gotoxy(10,10);
ok := false;
x := 61;
y := 10;
write(' SET VARIABLE(S) TO ZERO Y/N? ');
CHECK;
if answer[1] in ['Y','y'] then begin {5}
  gotoxy(23,12);
  writeln('ENTER VARIABLE NUMBER THEN RETURN ');
  gotoxy(21,14);
  writeln('END WITH NO NUMBER AND CARRIAGE RETURN ');
  writeln(' ');
  repeat
    write(' ');
    x := wherex;
    y := wherey;
    read(numbers);
  if length(numbers) <> 0 then begin {6}
    VAL(numbers,result,code);
    if (code=0) and ((result>0) and (result <= num_var+l)) then
      for i := 1 to num_cons+1 do
        AMATRIX[i,result+1] := 0.0
    else ERROR(x,y,1,0,z);
  end; {6}
  writeln(' ');
end; {5}
end; {1}
{***************************************************************************}
PROCEDURE Info;
var x3,y3,j : integer;
  answer : string[2];
  ok : boolean;
{--------------------------------INNER PROCEDURE--------------------------------}
PROCEDURE Checkit (var test:boolean);
var ok : boolean;
begin
ok := false;
in_info := false;
repeat
readln(answer);
if length(answer) <> 0 then begin
if (answer[1] in ['Y','y','N','n']) and
(length(answer)<2) then
if answer[1] in ['N','n'] then begin {1}
test := false;
ok := true;
end {1}
else begin {2}
test := true;
ok := true;
end {2}
else ERROR(x3,y3,1,0,j);
end
else ok := true;
until ok;
end; {checkit}

begin {info}
{PROBLEM SET UP}
clrscr;
done := false;
pagex := 1;
pagey := 1;
gotoxy(10,5);
write('MAXIMIZATION PROBLEM? Y)es or N)o
if minmax= true then write('Y') else write('N');
x3 := 61;
y3 := 5;
gotoxy(x3,y3);
CHECKIT(minmax);
gotoxy(10,7);
write('ENTER NUMBER OF STRUCTURAL VARIABLES ');
write(num_var);
x3 := 61;
y3 := 7;
gotoxy(x3,y3);
ERROR(x3,y3,2,25,j);
if j <> 0 then num_var := j;
gotoxy(10,9);
write('ENTER NUMBER OF CONSTRAINTS ');
write(num_cons);
x3 := 61;
y3 := 9;
gotoxy(x3,y3);
ERROR(x3,y3,2,50,j);
if j <> 0 then num_cons := j;
gotoxy(l0,11);
write(' ENTER PROBLEM NAME ');
readln(name);
if length(name)=0 then name := 'PROBLEM';
gotoxy(l0,13);
write(' RIGHT HAND SIDE RANGE ANALYSIS Y)es or N)o ');
if range_yes=true then write('Y') else write('N');
x3 := 61;
y3 := 13;
gotoxy(x3,y3);
CHECKIT(range_yes);
gotoxy(l0,15);
write(' DO YOU WANT INTERMEDIATE RESULTS Y)es or N)o ');
if intermediate=true then write('Y') else write('N');
x3 := 61;
y3 := 15;
gotoxy(x3,y3);
CHECKIT(intermediate);
in_info := true;
if intermediate=true then begin
  clrscr;
  repeat
    write(' DO YOU WANT INTERMEDIATE RESULTS ON PRINTER OR SCREEN? P/S ');
    readln(int_device);
    if int_device in ['P','p','S','s'] then ok := true
    else begin
      write(chr(7));
      gotoxy(10,1);
    end;
    until ok;
  end;
end; {info}
{**********************************************************************}
PROCEDURE Problem_in;
begin
  var xl,x2,y1,y2 : integer;
  i,j,k : integer;
  in_done : boolean;
  F : integer;
begin {l}
  repeat
    clrscr;
    gotoxy(3,4);
    write('Z = ');
    obj := false;
    GRID(obj);
    xl := 26;
    x2 := 51;
    y1 := 1;
    y2 := 2;
    COLOR(x1,x2,y1,y2);
xl := 1;
x2 := 80;
y1 := 21;
y2 := 23;
COLOR(x1, x2, y1, y2);
gotoxy(27, 1);
writeln('ENTER OBJECTIVE FUNCTION');
gotoxy(3, 22);
writeln('F1: ACCEPT LINE/GRID
F3: PAGE RIGHT');
gotoxy(3, 23);
writeln('F4: PAGE LEFT
F6: PAGE UP');
PAGE_OUT;
xl := 9;
x2 := 56;
y1 := 4;
y2 := 6;
F := 0;
MATRIX_IN(x1, x2, y1, y2, F);
PAGE_IN;
decimal := 0;
if F = 59 then in_done := true
else
  CHANGE_PAGE(F);
until in_done = true;
{end objective input}
pagex := 1;
pagey := 1;
in_done := false;
repeat
  xl := 30;
x2 := 48;
y1 := 1;
y2 := 2;
COLOR(x1, x2, y1, y2);
gotoxy(31, 1);
writeln('ENTER CONSTRAINTS');
xl := 1;
x2 := 80;
y1 := 21;
y2 := 23;
COLOR(x1, x2, y1, y2);
gotoxy(3, 22);
writeln('F1: ACCEPT LINE/GRID
F3: PAGE RIGHT');
gotoxy(3, 23);
writeln('F4: PAGE LEFT
F6: PAGE UP');
obj := true;
GRID(obj);
PAGE_OUT;
xl := 9;
\[ x_2 := 76; \]
\[ y_1 := 4; \]
\[ y_2 := 20; \]
\[ F := 0; \]
\[ \text{MATRIX\_IN}(x_1, x_2, y_1, y_2, F); \]
\[ \text{PAGE\_IN}; \]
\[ \text{decimal} := 0; \]
\[ \text{if } F = 59 \text{ then } \text{in\_done} := \text{true} \]
\[ \text{else} \]
\[ \text{CHANGE\_PAGE}(F); \]
\[ \text{until } \text{in\_done} = \text{true}; \]
\[ \text{end}; \] {1}

{*********************************************************}

\textbf{PROCEDURE} \text{Init\_basis};
\begin{verbatim}
var i, j, k : integer;
begin {1}
m := num\_cons + 1;
n := num\_var + 1;
equal := 0;
{CHANGE = TO < AND >}
for i := 2 to m do
if sign[i] = 0 then
begin {2}
equal := equal + 1;
for j := 1 to n do
 AMATRIX[m + equal, j] := AMATRIX[i, j];
SIGN[i] := -1;
SIGN[m + equal] := 1;
RHS[m + equal] := RHS[i];
end; {2}
m := m + equal;
for i := 2 to m do
if SIGN[i] = 1 then
begin {5}
for j := 2 to n do
 AMATRIX[i, j] := -1 * AMATRIX[i, j];
RHS[i] := -1 * RHS[i];
end; {5}
k := n + 1;
for i := 2 to m do
begin {6}
k := k + 1;
 AMATRIX[i, k] := 1;
end; {6}
for i := 1 to m do
for j := 1 to m do
if i = j then BINVERSE[i, j] := 1;
k := n + 1;
BASIS[1] := 1;
for i := 2 to m do
begin {3}

\end{verbatim}
\end{verbatim}
\begin{verbatim}
BASIS[i] := k;
k := k + 1;
XBI[i] := RHS[i];
end; {3}
for i := 1 to n-l do
NONBAS IS[i] := i+1;
n := n + m - 1;
end; {1}

************ PROCEDURE Inter_results (z:real); var i, j : integer;
temp : real;
wait : char;
begin
clrscr;
gotoxy(10,1);
writeln(' ');
if int_device in ['P', 'p'] then assign(filel, '1st:') else
assign(filel, 'con:');
reset(filel);
if int_device in ['P', 'p'] then
   writeln(' PRINTING INTERMEDIATE
RESULTS');
   writeln(filel,' ',name);
   writeln(filel,' INTERMEDIATE
RESULTS');
   writeln(filel,' ');
   writeln(filel,' ',round(iteration));
   writeln(filel,' ');
   writeln(filel,' LEAVE
VECTOR IS
X ( ',NONBAS IS[-1,'] ') ');
   writeln(filel,' ');
   writeln(filel,' ENTER
VECTOR IS
X('',BASIS[-1,'] ')');
   writeln(filel,' ');
   writeln(filel, 'OLD Z VALUE IS
:', z);
   writeln(filel,' ');
   writeln(filel, 'NEW Z VALUE IS
:', XBI[1]);
   writeln(filel,' ');
   writeln(filel, 'CHANGE IN
Z IS
:', XBI[1] - z);
   writeln(filel,' ');
if int_device in ['P', 'p'] then
   for i := 1 to 5 do writeln(filel,' ')
else begin
   gotoxy(28,20);
   write('PRESS ENTER WHEN READY');
   readln(wait);
end;
end;
end;
\end{verbatim}
end;
end;

{*******************************************************************************
PROCEDURE Matrix_mult (var A : matrix; var D,B : single;
acol,begin_row,end_row : integer);

var ii,kk : integer;
begin {1}
B := CLEAR_SIN;
for ii := begin_row to end_row do
begin {2}
B[ii] := 0;
for kk := 1 to acol do
end; {2}
end; {1}
{*******************************************************************************
PROCEDURE Scalar_mul (scalar : real; var D,E : single;
arow : integer);

var is : integer;
begin {1}
D := CLEAR_SIN;
for is := 1 to arow do
D[is] := scalar * E[is];
end; {1}

{*******************************************************************************
PROCEDURE Add_matrix (var B,D,E : single; arow : integer);

var ia : integer;
begin {1}
B := CLEAR_SIN;
for ia := 1 to arow do
B[ia] := D[ia] + E[ia];
end; {1}

{*******************************************************************************
PROCEDURE Feasible;
var kf, fi, min : integer;
    negative : boolean;
    hold : isingle;
begin {1}
remove := Ø;
negative := false;
kf := 1;
for fi := 2 to m do
if XBI[fi] < Ø then
begin {2}
negative := true;
hold[kf] := fi;
kf := kf + 1;
end; {2}
if negative = true then
begin {3}
min := hold[1];
for fi := 1 to (kf-1) do
  if XBI[min] > XBI[hold[fi]] then min := hold[fi];
  remove := min;
  route := 1;
  end {3}
else
  route := 0;
end; {l}

PROCEDURE Zcj;
var DD,zdummy : single;
iz,jz,kz : integer;
count integer;
begin {l}
  count := 0;
  DD := CLEAR_SIN;
  for iz := 1 to (n-m) do
    begin {2}
      for jz := 1 to m do
        DD[jz] := AMATRIX[jz,NONBASIS[iz]];
        MATRIX_MULT(BINVERSE,DD,zdummy,m,1,1);
        ZJCJ[iz] := zdummy[1];
      end; {2}
      if ((minmax=true) and (ZJCJ[iz]<0)) or ((minmax=false) and (ZJCJ[iz]>0)) then count:=l;
    end; {2}
  if count <> 1 then done := true else done := false;
end; {l}

PROCEDURE Primal_enter;
var zero : real;
i : integer;
begin {l}
  zero := 0;
  for i := 1 to (n-m) do
    if ((minmax=true) and (ZJCJ[i] < zero)) or ((minmax=false) and (ZJCJ[i] > zero)) then
      begin {2}
        zero := ZJCJ[i];
        enter := i;
      end; {2}
end; {l}

PROCEDURE Ycol;
var i,j : integer;
begin {l}
  DUMMY := CLEAR_SIN;
  for j := 1 to m do
    DUMMY[j] := AMATRIX[j,NONBASIS[enter]];
    MATRIX_MULT(BINVERSE,DUMMY,YK,m,1,m);
    num_test := 0;
    for i := 2 to m do
      if YK[i] > 0 then begin {2}
num_test := num_test + 1;
INDEX_TEST[num_test] := i;
end; {2}
if num_test<>0 then nosoln:=false else nosoln := true;
end; {1}

{**********************************************************************}
PROCEDURE Primal_remove;
var select_old, select_new : real;
hold : isingle;
i : integer;
begin {1}
select_old := XBI[INDEX_TEST[l]]/YK[INDEX_TEST[l]];
remove := INDEX_TEST[l];
for i := 2 to num_test do
begin {2}
select_new := XBI[INDEX_TEST[i]]/YK[INDEX_TEST[i]];
if select_new < select_old then begin {3}
remove := INDEX_TEST[i];
select_old := select_new;
end; {3}
end; {2}
end; {1}

{**********************************************************************}
PROCEDURE Transform;
var drj, pivot : real;
nk, newcol, colt : single;
change, i, j : integer;
begin {1}
iteration := iteration + 1;
if route = 1 then
YCOL;
pivot := YK[remove];
nk[l] := -YK[l]/pivot;
for i := 2 to m do
if i <> remove then nk[i] := -YK[i]/pivot
else nk[i] := (I/pivot) - 1;
for j := 2 to m do
begin {2}
drj := BINVERSE[remove,j];
SCALAR_MUL(drj,DUMMY,nk,m);
for i := 1 to m do
colt[i] := BINVERSE[i,j];
ADD_MATRIX(newcol, colt, DUMMY, m);
for i := 1 to m do
BINVERSE[i,j] := newcol[i];
end; {2}
drj := XBI[remove];
SCALAR_MUL(drj,DUMMY,nk,m);
for i := 1 to m do
colt[i] := XBI[i];
ADD_MATRIX(XBI, colt, DUMMY, m);
change := BASIS[remove];
BASIS[remove] := NONBASIS[enter];
NONBASIS[enter] := change;
end;  \{1\}
{*******************************************************
PROCEDURE  Dual_remove;
var   id  : integer;
      xmin : real;
begin  \{1\}
xmin := XBI[2];
remove := 2;
for id := 3 to m do
if XBI[id] < xmin then
  begin \{2\}
    xmin := XBI[id];  
    remove := id;
  end; \{2\}
end; \{1\}
{*******************************************************
PROCEDURE  Yrow;
var  iz, iy  : integer;
      hold,holdl : single;
begin \{1\}
num_test := 0;
for iy := 1 to (n-rn) do
  begin \{2\}
    for iz := 1 to m do
      hold[iz] := AMATRIX[iz,NONBASIS[iy]]; 
    for iz := 1 to m do
      MATRIX_MULT(BINVERSE,hold,holdl,m,remove,remove);
    YRJ[iy] := holdl[remove];
    if YRJ[iy] < 0 then
      begin \{3\}
        num_test := num_test + 1;
        INDEX_TEST[num_test] := iy;
      end; \{3\}
  end; \{2\}
if num_test <> 0 then nosoln:=false else nosoln:= true;
end; \{1\}
{*******************************************************
PROCEDURE  Dual_enter;
var  is : integer;
      select_old,select_new : real;
begin \{1\}
select_old := ZJJCJ[INDEX_TEST[1]]/YRJ[INDEX_TEST[1]];
enter := INDEX_TEST[1];
for is := 2 to num_test do
  begin \{2\}
    select_new := ZJJCJ[INDEX_TEST[is]]/YRJ[INDEX_TEST[is]];
    if ((minmax=true) and (select_new>select_old)) or
      ((minmax=false) and (select_new<select_old)) then
      begin \{3\}
        select_old := select_new;
      end; \{3\}
    end; \{2\}
enter := INDEX_TEST[i];
end; {3}
end; {2}
end; {1}

{**************************************************************************}

PROCEDURE Analysis;
var
  k, i, j : integer;
  dik, upper, lower, delta1, delta2 : real;
  countl, count2, temp : integer;
  change : integer;
begin {analysis}
  for i := 2 to m do begin {1}
    countl := 0;
    count2 := 0;
    upper := 0;
    lower := 0;
    for j := 2 to m do begin {2}
      dik := BINVERSE[j,i];
      if abs(dik) < roundoff then dik := 0;
      if (dik < 0) then
        if ((XBI[j]=0) and (BASIS[j]>num_var+1)) then k := 0 else
          case countl of
            0 : begin {3}
              upper := -XBI[j]/dik;
              count1 := 1;
            end; {3}
            else begin {4}
              delta1 := -XBI[j]/dik;
              if upper > delta1 then upper := delta1;
            end; {4}
          end; {case}
        if (dik > 0) then
          if ((XBI[j]=0) and (BASIS[j]>num_var+1)) then k := 0 else
            case count2 of
              0 : begin {5}
                lower := -XBI[j]/dik;
                count2 := 1;
              end; {5}
              else begin {6}
                delta2 := -XBI[j]/dik;
                if lower < delta2 then lower := delta2;
              end; {6}
            end; {case}
          end; {2}
        case SIGN[i] of
          1 : begin
            RANGE[1,i] := -1 * (RHS[i] + lower);
            RANGE[2,i] := -1 * (RHS[i] + upper);
            RANGE[3,i] := count2;
            RANGE[4,i] := count1;
          end
    end;
end;
else begin
    RANGE[1,i] := RHS[i] + upper;
    RANGE[2,i] := RHS[i] + lower;
    RANGE[3,i] := count1;
    RANGE[4,i] := count2;
end;
end; {case}
end; {1}
for i := 2 to num_cons+1 do
    if RELATIONS[i] = Ø then begin {7} {test if =}
        count2 := num_cons + i;
        temp := round(RANGE[4,i]);
        case temp of
        Ø : if RANGE[4,count2] <> Ø then begin {8}
            RANGE[4,i] := 1;
            RANGE[2,i] := RANGE[2,count2];
            end; {8}
        1 : if RANGE[4,count2] <> Ø then
            if RANGE[2,count2] > RANGE[2,i] then
                RANGE[2,i] := RANGE[2,count2];
            end; {case}
        temp := round(RANGE[3,i]);
        case temp of
        Ø : if RANGE[3,count2] <> Ø then begin {9}
            RANGE[3,i] := 1;
            RANGE[1,i] := RANGE[1,count2];
            end; {9}
        1 : if RANGE[3,count2] <> Ø then
            if RANGE[1,count2] < RANGE[1,i] then
                RANGE[1,i] := RANGE[1,count2];
            end; {case}
        end; {7}
    end; {analysis}
{***************************************************************************************************************
PROCEDURE Solve;
var isover,over : boolean;
z_old : real;
wait : char;
begin {solve}
crlscr;
gotoxy(36,1);
writeln('SOLVING');
GETTIME(cx2,dx2);
MAKETIME(timel);
repeat
    isover := false;
    FEASIBLE;
    if route = Ø then begin {12}
        {PRIMAL}
    begin {1}
        repeat
            over := false;
if done = true then over := true
else begin {3}
    PRIMAL_ENTER;
    YCOL;
    if nosoln=false then begin {4}
        PRIMAL_REMOVE;
        z_old := XBI[1];
        TRANSFORM;
        if intermediate=true then INTER_RESULTS(z_old);
        if iteration > it_limit then over := true;
        end {4}
    else begin {5}
        over := true;
        clrscr;
        gotoxy(21,1);
        writeln('PRIMAL UNBOUNDED ===> ');
        if minmax=true then writeln(' NO UPPER BOUND') else writeln(' NO LOWER BOUND');
        gotoxy(19,3);
        writeln('SELECT OUTPUT TO SEE LAST STATE OF PROBLEM');
        gotoxy(29,5);
        writeln('PRESS ENTER WHEN READY');
        readln(wait);
    end; {5}
    done := true;
end; {3}
until over;
end; {1}
isover := true;
end {12}
{DUAL}
else begin {11}
    DUAL_REMOVE;
    YROW;
    if nosoln=false then begin {6}
        ZCJ;
        DUAL_ENTER;
        z_old := XBI[1];
        TRANSFORM;
        if iteration > it_limit then isover := true;
        end {6}
    else begin {7}
        isover := true;
        clrscr;
        gotoxy(20,1);
        writeln('DUAL UNBOUNDED ===> NO FEASIBLE SOLUTION ');
        gotoxy(19,3);
        writeln('SELECT OUTPUT TO SEE LAST STATE OF PROBLEM');
        gotoxy(29,5);
        writeln('PRESS ENTER WHEN READY');
        readln(wait);
    end; {7}
    }
end; {7}
if intermediate=true then INTER_RESULTS(z_old);
end; {11}
until isover;
GETTIME(cx2,dx2);
ELAPSED(time,cx1,dx1,cx2,dx2);
MAKETIME(time2);
end; {solve}

{**********************************************************}
PROCEDURE Print_out;
var
templ,temp2,temp3 : real;
i,j : integer;
multiplier,line_count : integer;
relation: string[2];
counter : boolean;
start,finish : integer;
wait : char;

{------------------INNER PROCEDURES------------------------}
PROCEDURE Screen_hold (lines:integer);
begin
if line_count > lines then begin
    line_count := 0;
    writeln(' ');
    gotoxy(1,wherey+3);
write(' PRESS ENTER WHEN READY');
readln(wait);
clrscr;
end;
end;
PROCEDURE Section_1;
begin {section_1}
gotoxy(36,1);
writeln('PRINTING');
if printer=false then clrscr;
writeln(filel,'NAME: ',name);
if minmax= true then write(filel,'TYPE: MAXIMIZATION ')
else write(filel,'TYPE: MINIMIZATION ');
writeln(filel,'ITERATIONS: ',iteration:6:0);
line_count := 2;
if (nosoln=true) or (iteration > it_limit) then begin
    writeln(filel,' NO SOLUTION');
    writeln(filel,' ');
    writeln(filel,'VALUES FOR LAST STATE OF PROBLEM');
    writeln(filel,' ');
    line_count := 6;
end;
if print_old = false then begin
for i := 1 to 4 do writeln(filel,' ');
end;
writeln(filel,'
TIME ','time1);
writeln(filel,'
TIME ','time2);
writeln(filel,'
line_count := line_count + 9;
end;
for i := 1 to 4 do writeln(filel,'
line_count := line_count + 4;
counter := false;
start := 2;
finish := 5;
repeat {objective function}
if num_var < finish then finish := num_var;
for i := start to finish do begin
  templ := -l*AMATRIX[1,i];
  if i=2 then
    write(filel,'Z = ',templ:fore:aft,'X(',i-1,') + ')
  else
    if start=2 then write(filel,templ:fore:aft,'X(',i-1,') + ')
    else write(filel,' ','templ:fore:aft,'X( ',i-1, ') + ')
end;
  templ:=-l*AMATRIX[1,finish+1];
  writeln(filel,templ:fore:aft,'X(','finish,') ');
  line_count := line_count + 1;
if finish+1=num_var+1 then counter:=true
else begin
  start := finish+2;
  finish := finish+5;
  end;
until counter;
writeln(filel,'
writeln(filel,'
line_count := line_count + 2;
counter := false;
start := 2;
finish := 5;
repeat {constraints}
if num_var < finish then finish := num_var;
for i := 2 to num_cons+1 do
begin {2}
  multiplier := l;
case RELATIONS[i] of
    -1 : relation := '<=';
    0 : relation := '=';
    1 begin
      relation := '>=';
      if old=false then multiplier := -1;
    end;
  end; {case}
if printer=false then SCREEN_HOLD(18);
for j := start to finish do begin
templ := multiplier*AMATRIX[i,j];
if j=start then write(filel,'
',templ:fore:aft,'X(',j-1,'')+') else
   write(filel,templ:fore:aft,'X(',j-1,'')+');
end;

if finish+1=num_var+1 then writeln(filel,relation,'n',multiplier*RHS[i]:fore:aft);
else writeln(filel,',');
line_count := line_count+1;
end;

if finish+1=num_var+1 then counter:=true
else begin
   start := finish+2;
   finish := finish + 5;
   end;
if printer=false then SCREEN_HOLD(18);
writeln(filel,' ');
line_count := line_count + 1;
until counter;
end; {section_1}

PROCEDURE Section_2;
begin {section_2}
for i := 1 to 4 do writeln(filel,' ');

writeln(filel,' VARIABLE I)
---------');

if m > (n-m) then j:= m else j:=(n-m);
for i := 1 to j do begin {a}
   if i=1 then writeln(filel,' BASIS:
X(',BASIS[i+1]-1,')
NO NBAS IS: X(',NONBASIS[i]-1,')');
else begin {b}
   if (i+1 <= m) and (i <= (n-m)) then
      writeln(filel,' X(',BASIS[i+1]-1,')
X(',BASIS[i+1]-1,')');
else begin {c}
   if (i+1 <= m) then writeln(filel,' X(',BASIS[i+1]-1,')');
   else if (i <= (n-m)) then
      writeln(filel,' X(',BASIS[i+1]-1,')');
   end; {c}
end; {b}
   line_count := line_count + 1;
   if printer=false then SCREEN_HOLD(18);
end; {a}
end; {section_2}
PROCEDURE section_3;
begin {section_3}
for i := 1 to 4 do writeln(filel,' ');
templ := XBI[1];
writeln(filel,' SOLUTION IS');
writeln(filel,'------------');
writeln(filel,' z = ',templ:fore:aft);
line_count := 8;
for i := 2 to num do begin
if XBI[i] < roundoff then templ := 0.0 else
templ := XBI[i];
writeln(filel,' X(',BASIS[i]-1,') = ',templ:fore:aft);
line_count := line_count + 1;
if printer=false then SCREEN_HOLD(18);
end;
end; {section_3}
PROCEDURE Section_4;
begin {section_4}
for i := 1 to 4 do writeln(filel,' ');
writeln(filel,' RIGHT BAND SENSITIVITY ANALYSIS');
writeln(filel,'-------------------------------');
writeln(filel,' ');
line_count := 7;
for i := 2 to num_cons+1 do begin
templ := RANGE[2,i]; {lower}
temp2 := RANGE[1,i]; {upper}
if (RELATIONS[i]=1) and (print_old=false) then temp3 := -1*RHS[i] else temp3 := RHS[i];
write (filel,' ');
if RANGE[4,i]=0 then write(filel,'NO LOWER') else
write(filel,temp1:fore:aft);
write(filel,' <= ',temp3:fore:aft,' <= ');
if RANGE[3,i]=0 then writeln(filel,'NO UPPER') else
writeln(filel,temp2:fore:aft);
line_count := line_count + 1;
if printer = false then SCREEN_HOLD(18);
end;
end; {section_4}
if printer=false then begin
  line_count := 19;
  SCREEN_HOLD(18);
  line_count := 0;
  clrscr;
end;
SECTION_2;
if printer = false then begin
  line_count := 19;
  SCREEN_HOLD(18);
  line_count := 0;
  clrscr;
end;
SECTION_3;
if printer = false then begin
  line_count := 19;
  SCREEN_HOLD(18);
  line_count := 0;
  clrscr;
end;
if range_yes = true then begin
  SECTION_4;
  if printer = false then begin
    line_count := 19;
    SCREEN_HOLD(18);
    line_count := 0;
    clrscr;
  end;
end;
end; {print_out}

{**********************************************************}
PROCEDURE Disk_write;
var
  i, j : integer;
  templ, temp2 : real;
begin {disk_write}
gotoxy(10,1);
  WRITING ',name,' TO DISK ');
  writeln(';
  assign(file2,name);
  rewrite(file2); {open to write to}
  if nosoln=false then templ := 1.0 else templ := 0.0;
  write(file2,templ);
  if minmax=true then templ := 1.0 else templ := 0.0;
  write(file2,templ);
  templ := num_cons;
  write(file2,templ);
  templ := num_var;
  write(file2,templ);
  templ := m;
  write(file2,templ);
  templ := n;
  write(file2,templ);
  write(file2,iteration);
  templ := fore;
write(file2,templ);
templ := aft;
write(file2,templ);
for i := 1 to num_cons+l do {objective & constraints}
    for j := 1 to num_var+l do begin
        templ := AMATRIX[i,j];
        if RELATIONS[i]=l then templ := -l*templ;
        write(file2,templ);
    end; {constraints}
for i := 1 to num_cons+l do begin {signs}
    templ := RELATIONS[i];
    write(file2,templ);
end; {signs}
for i := 1 to num_cons+l do begin {rhs}
    templ := RHS[i];
    if RELATIONS[i]=l then templ := -l*templ;
    write(file2,templ);
end; {rhs}
for i := 1 to m do begin {basis}
    templ := BASIS[i];
    write(file2,templ);
end; {basis}
for i := 1 to (n-m) do begin {nonbasis}
    templ := NONBASIS[i];
    write(file2,templ);
end; {nonbasis}
for i := 1 to m do begin {xbi solution}
    templ := XBI[i];
    write(file2,templ);
end; {solution}
if range_yes=true then begin {analysis}
    templ := 1.0;
    write(file2,templ);
    for i := 1 to num_cons+l do begin
        templ := RANGE[2,i];
        temp2 := RANGE[1,i];
        write(file2,templ,temp2);
    end;
end
else begin
    templ := 0.0;
    write(file2,templ);
end;
close(file2);
end; {disk_write}
{*******************************************************************************}
PROCEDURE Disk_read;
var i,j : integer;
templ,temp2 : real;
begin {disk_read}
if file_check = true then begin {read}
first_time := false;
read(file2,templ);
if temp=1.0 then nosoln := false else nosoln := true;
read(file2,templ);
if temp=1.0 then minmax := true else minmax := false;
read(file2,templ);
num_cons := round(templ);
read(file2,templ);
um_var := round(templ);
read(file2,templ);
m := round(templ);
n := round(templ);
read(file2,iteration);
read(file2,templ);
fore := round(templ);
read(file2,templ);
aft := round(templ);
for i := 1 to num_cons+1 do {objective & constraints}
for j := 1 to num_var+1 do begin
read(file2,templ);
AMATRIX[i,j] := templ;
end;
for i := 1 to num_cons+1 do begin {signs}
read(file2,templ);
RELATIONS[i] := round(templ);
end;
for i := 1 to num_cons+1 do begin {rhs}
read(file2,templ);
RHS[i] := templ;
end;
for i := 1 to m do begin {basis}
read(file2,templ);
BASIS[i] := round(templ);
end;
for i := 1 to (n-m) do begin {nonbasis}
read(file2,templ);
NONBASIS[i] := round(templ);
end;
for i := 1 to m do begin {xbi solution}
read(file2,templ);
XB[i] := templ;
end;
read(file2,templ);
if temp=1.0 then range_yes := true else range_yes := false;
if range_yes=true then
for i := 1 to num_cons+1 do begin
read(file2,templ,temp2);
RANGE[2,i] := templ;
RANGE[1,i] := temp2;
end;
end;
read(file2,temp1,temp2);
RANGE[4,i] := temp1;
RANGE[3,i] := temp2;
end;
close(file2);
end; {read}
end; {disk_read}
PROCEDURE Menu (st1,st2,st3,st4 : cue;
choice : integer);
var   key1 : char;
mover : boolean;
i : integer;
begin {menu}
clrscr;
for i:= 1 to 6 do writeln(' ');
writeln(messages[1]);
writeln(' ');
writeln(st1);
writeln(st2);
writeln(st3);
writeln(st4);
repeat
read(kbd,key1);
if ord(key1)=27 then begin {2}
read(kbd,key1);
case ord(key1) of
59 : begin
   case choice of
     0 : selection := 0;
     1 : selection := 3;
     2 : selection := 6;
     3 : selection := 7;
   end; {case}
mover := true;
clrscr;
end;
60 : begin
   case choice of
     0 : begin
       if first_time = true then begin
         clrscr;
gotoxy(l0,1);
writeln(' THERE IS NO PROBLEM TO OUTPUT');
goxy(l0,3);
writeln(' SELECT AGAIN');
goxy(l0,5);
write(' ENTER WHEN READY');
readln(key1);
end;
   end;
end;
end; {Menu}
es[4], messages[15], 0);

MENU(messages[2], messages[3], messages[4],

end
else selection := 1;
end;

1 : selection := 4;
2 : selection := 10;
3 : selection := 8;
end; {case}
mover := true;
clrscr;
end;

61 : begin
case choice of
  0 : selection := 2;
  1 : selection := 5;
  2 : selection := 5;
  3 : selection := 9;
end; {case}
mover := true;
clrscr;
end;

62 : begin
selection := 5;
mover := true;
clrscr;
end;

else begin {3}
writeln(chr(7));
mover := false;
end; {3}
end; {case}
end {2}
else begin {4}
writeln(chr(7));
mover := false;
end; {4}
until mover;
end; {menu}

{**********************************************************}
PROCEDURE Logo;
var wait : char;
begin
clrscr;
gotoxy(38, 6);
writeln('RDS');
gotoxy(30, 8);
writeln('SOFTWARE PACKAGE FOR');
gotoxy(31, 10);
writeln('LINEAR PROGRAMMING');
gotoxy(27, 17);
writeln('Copyright 1985 Debbie Fogal');

PROCEDURE Logo;
gotoxy(30,18);
writeln('all rights reserved');
gotoxy(29,20);
write('PRESS ENTER WHEN READY');
readln(wait);
end;
{======================== = == = = ============================}
{MAIN}
var    finished : boolean;
       wait : char;
begin   {1}
   clrscr;
   finished := false;
   first_time := true;
   MESSAGE;
   TEXTCOLOR(15);
   TEXTBACKGROUND(0);
   LOGO;
   MENU(messages[2],messages[3],messages[4],messages[15],0);
repeat
   case selection of
   0 : begin  {input problem type}
      old := false;
      INIT;
      MENU(messages[5],messages[6],messages[7],messages[15],1);
      end;
   1 : begin  {output device selection}
      MENU(messages[10],messages[11],messages[12],messages[13],3);
      end;
   2 : begin  {exit program}
      clrscr;
      write('EXIT PROGRAM? Y/N ');
      readln(wait);
      if wait in ['Y','y'] then finished := true
      else
      MENU(messages[2],messages[3],messages[4],messages[15],0);
      end;
   3 : begin  {input new}
      old := false;
      INIT;
      first_time := false;
      INFO;
      PROBLEM_IN;
      INIT_BASIS;
      clrscr;
      SOLVE;
      if iteration > it_limit then begin
         clrscr;
         gotoxy(18,1);
         writeln('NUMBER OF ITERATIONS EXCEEDS ITERATION LIMIT');
      end;
end;
PROBLEM');
go toxy (32,3);
writeln('WILL NOT PROCEED');
go toxy (19,5);
writeln('SE LECT OUTPUT TO SEE LAST STATE OF PROBLEM');
go toxy (29,7);
write('PRESS ENTER WHEN READY');
readln (wait);
end
else if range_yes=true then ANALYSIS;
MENU (messages[2], messages[3], messages[4], messages[15], 0);
end; {input new}
4 : begin {old problem}
old := true;
INIT;
go toxy (10,1);
write('ENTER DISK FILE NAME ');
readln (name);
assign (file2,name);
{$I-}
reset (file2);
{$I+}
file_check := (IOresult=0);
if file_check = false then begin
clrscr;
go toxy (28,1);
writeln ('FILE DOES NOT EXIST');
go toxy (28,3);
write('PRESS ENTER WHEN READY');
readln (wait);
selection := 5;
close (file2);
end
else begin
MENU (messages [8], messages [9], messages [7], messages [15], 2);
if selection <> 5 then DISK_READ;
end;
end;
5 : begin {main menu}
old := false;
MENU (messages [2], messages [3], messages [4], messages [15], 0);
end;
6 : begin {edit}
old := true;
INIT;
CHANGE;
INFO;
PROBLEM_IN;
INIT_BASIS;
SOLVE;
if iteration > it_limit then begin
clrscr;
gotoxy(18,1);

writeln('NUMBER OF ITERATIONS EXCEEDS ITERATION LIMIT');
gotoxy(32,3);
writeln('WILL NOT PROCEED');
gotoxy(19,5);
writeln('SELECT OUTPUT TO SEE LAST STATE OF PROBLEM');
gotoxy(29,7);
write('PRESS ENTER WHEN READY');
readln(wait);
end

else if range_yes=true then ANALYSIS;
old := false;
MENU(messages[2],messages[3],messages[4],messages[15],0);
end; {edit}

7 : begin {printer}
assign(file1,'lst:');
reset(file1);
printer := true;
clrscr;
gotoxy(33,10);
writeln('PREPARE PAPER');
gotoxy(32,12);
writeln('TURN ON PRINTER');
gotoxy(29,14);
write('PRESS ENTER WHEN READY ');
readln(wait);
clrscr;
PRINT_OUT;
MENU(messages[2],messages[3],messages[4],messages[15],0);
end; {printer}

8 : begin {screen}
assign(file1,'con:');
reset(file1);
printer := false;
PRINT_OUT;
MENU(messages[2],messages[3],messages[4],messages[15],0);
end; {screen}

9 : begin {disk}
DISK_WRITE;
MENU(messages[2],messages[3],messages[4],messages[15],0);
end;

10 : begin {output old}
old := true;
print_old := true;
MENU(messages[10],messages[11],messages[12],messages[13],3);
end; {case}
until finished;
end. {1}
REFERENCES


