This thesis presents a linear programming package that uses an algorithm and data storage method efficient enough to solve a medium sized linear programming problem (250 variables and 100 constraints) on a minicomputer with small core memory space (4000 bytes).

In this routine the problem is formulated by the use of Thompson's method and the numbers are stored using the condensed tableau formulation. The product form of the inverse is used within the computational algorithm.

Another objective of the thesis is to design an input-output system which presents the linear programming model in a form that is compatible with Resource Planning and Management System networks.

The minicomputer that is used to validate these objectives is a Wang 2200 with 4 k-bytes of CPU memory and a microprogrammed
BASIC language. The features of Wang 2200 that are utilized in this study include variable word size, BIT manipulation, and packing of information.

The linear programming package presented in this thesis proved that a BASIC program can be written for a 4 k-bytes mini-computer to solve RPM problems with as many as 250 process and 100 resource nodes.

Though the computational speed was found to be greatly hindered by the physical speed of data transfer to and from the auxiliary memory (cassette tapes) device, the cost of computation was found to be comparable to the other mathematical programming software packages on CDC 3300 and CDC CYBER-73.
Implementation of Revised Simplex Thompson Algorithm on a Limited Core Stand Alone Processor

by

Jyotirmoy Chakravarty

A THESIS

submitted to

Oregon State University

in partial fulfillment of the requirements for the degree of

Master of Science

Completed June 1976

Commencement June 1978
TO BERNADETTE

in recognition of her unfathomable patience
ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Dr. Michael S. Inoue for his constant encouragement and guidance during the course of this thesis.

I also wish to acknowledge my friends and colleagues for their suggestions and assistance. In particular I wish to recognize Mr. R. Hannah and Mr. J. Rousey of U.S. Forest Service for their expert opinions and for letting me use the WANG 2200 facilities in their office. I also wish to thank Mr. R. Radhakrishnan for his critical review of this thesis.

Finally, I wish to thank Mrs. M.J. Stratton for doing a great job of typing the manuscript under tremendous time constraints.
TABLE OF CONTENTS

I. INTRODUCTION

Research Objective
Structure of the Thesis

II. RESOURCE PLANNING AND MANAGEMENT SYSTEM (RPMS)

Components of RPMS
Postulates of RPMS
RPMS Conventions
Construction of the RPMS Network
Advantages of RPMS Networks

III. A REVIEW OF THE GENERAL LINEAR PROGRAMMING ALGORITHMS

Definition of Linear Programming
General Forms of Linear Programming Problems
Concept of a Basis
A Survey of the Algorithms
Product Form of Inverse
Thompson's Method

IV. LPRPM ALGORITHM

Condensed Tableau Method

V. CORE SPACE ALLOCATION ON WANG 2200

Program Storage
Data Storage
Pointer System

VI. EVALUATIONS AND RECOMMENDATIONS

Computational Performance
Built-in Debugging Aids
Recommendations for Future Studies

BIBLIOGRAPHY

APPENDICES
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>System flowchart.</td>
<td>6</td>
</tr>
<tr>
<td>2-1</td>
<td>RPMS nodal conventions.</td>
<td>15</td>
</tr>
<tr>
<td>2-2</td>
<td>RPMS basic flow</td>
<td>17</td>
</tr>
<tr>
<td>2-3</td>
<td>RPMS cause and effect diagrams</td>
<td>17</td>
</tr>
<tr>
<td>2-4</td>
<td>Minimizing and maximizing nodes</td>
<td>18</td>
</tr>
<tr>
<td>2-5</td>
<td>RPMS feasibility and optimality conditions.</td>
<td>20</td>
</tr>
<tr>
<td>3-1</td>
<td>Simplex polyhedron</td>
<td>28</td>
</tr>
</tbody>
</table>
IMPLEMENTATION OF REVISED SIMPLEX THOMPSON ALGORITHM ON A LIMITED CORE STAND ALONE PROCESSOR

I. INTRODUCTION

The use of specialists conversant in Operations Research and related analytical techniques for management decision making within larger companies is receiving increasing acceptance. According to a survey conducted by J. K. Gerdel in 1969, the percentage of managerial and administrative personnel has remained relatively stable since 1940; however, the study points out that persons with a quantitative background such as economics, mathematics, statistics, operations research and related analytical and logical disciplines are being employed increasingly by businesses to aid management in decision making. A study conducted in 1975 showed that 36% of all the Operations Research projects in 1974 were funded by private industries (Shannon, 1975).

Included in the realm of Operations Research are the more popular optimization and simulation techniques such as: Linear Programming (LP), Mixed Integer Programming, Dynamic Programming, Stochastic, Nonlinear Programming, and Monte Carlo Methods. Owing to their less complex nature and the availability of computer programs, Linear Programming and Mixed Integer
Programming are considered the most popular Operations Research tools (Barnes, 1975).

With the advent of high speed and relatively low costing minicomputers, more and more smaller companies are expanding their in-house computational capabilities. In a reader survey conducted in 1972 (Rand Report, 1973), 2,000 minicomputer users, which represented approximately 30% of the total users in the United States and Canada, estimated that they would purchase 12,015 additional minicomputers over a period of 12 months. This represents an increase of 125% in the overall number of minicomputers.

It is, however, interesting to note that these smaller companies with sales between 5 and 20 million dollars tend to utilize minicomputers for accounting and record-keeping and shy away from developing Operations Research or analytical models on their computers (Frick and Hoisington, 1970). Unfortunately, the mathematical jargon -- the general aura of mystery -- have frightened away many small-business managers from the use of Operations Research techniques. Those who do understand the basics of optimization and simulation often do not wish to apply them because they believe that a practical production scheduling problem (one with greater than 10 variables and 10 constraints) formulated by linear programming cannot be solved on the available minicomputers. Most of the time the cost involved in hiring the services of a larger computer system is not justifiable either.
There is no strict definition of what a minicomputer is. However, it is possible to describe the general characteristics of a minicomputer (Weitzman, 1974):

1. Central processor is a single address device.
2. Word size is 8 to 18 bits.
3. Most of them have core memories with cycle time ranging from 600 nanoseconds to 1 microsecond.
4. The memory size limit for most is 4K (where 1K = approximately 1000 words), though some present-day mini's go up to 64K.
5. The price ranges from $3,000 to $20,000.

Some computing machines are found which are referred to as minicomputers but have exceptions and variations to these characteristics. The minicomputer industry is constantly evolving newer and faster machines; however, no major steps have been taken to improve the software packages that are available to solve Operations Research type problems. A survey done by the author showed that "Varian 73" minicomputer (Vanderweed) and PDP-11 of Digital Equipment Corporation do not have any linear programming packages available. IBM 5100 with 8K (bytes) memory (Senders, 1976) and Wang 2200 with 4K (bytes) memory (Wang Manual No. GLBR 22) have linear programming packages to handle small problems with up to 10 variables and 10 constraints.
Research Objective

The purpose of this thesis is two-fold:

1. To develop an algorithm and a software package that is efficient enough to solve linear programming problems with up to 250 variables and 100 constraints on minicomputers with small core memory space (4000 bytes). Also to validate the package using a Wang 2200 minicomputer.

2. To develop an input output format which would minimize the complexities of mathematical jargon in linear programming.

For the most efficient use of the core memory space the revised simplex method with some modifications along with Thompson's method of starting with a feasible solution have been used. The computation involved in each iteration are performed using the product form of the inverse method. The data are always stored in a very condensed form.

For the second goal, a visual, network approach developed by Inoue and Riggs in 1972 called Resource Planning and Management Systems is used. This deals with the relationships between scarce resources available to a production system and the utilization of these resources within the system. The software package is designed to conform to RPMS.
The software package developed in this study is called "LPRPM."

**Structure of the Thesis**

This introductory chapter discussed the need for software packages on minicomputers to solve Operations Research type problems. Chapter II introduces and explains Resource Planning and Management Systems techniques and their adaptation as an input/output format for the computer package in this research. Chapter III reviews the general forms of existing linear programming algorithms and their suitability to the problem in this thesis. Chapter IV describes the proposed approach along with a numerical example. Chapter V discusses the software limitations of Wang 2200 and the data storage techniques used in the proposed package. Chapter VI presents the evaluations and recommendations on the proposed study.

A flow chart of the LPRPM system is shown in Figure 1-1.
A. Instructions
B. Do you want to
   (1) Create new data file
   (2) Modify old data file
   (3) Run existing data
C. Input data to user cassette
D. Transfer data from user cassette to working cassette
E. Is solution
   (1) Infeasible
   (2) Feasible but non-optimal
   (3) Feasible and optimal
F. Determine pivot row, pivot column, pivot element and pivot vector
G. Compute new matrix
H. Output to printer

Figure 1-1. System flowchart.
II. RESOURCE PLANNING AND MANAGEMENT SYSTEM (RPMS)

An integral part of this thesis is Resource Planning and Management system. One of the purposes of this thesis is to simplify the understanding of linear programming to people who have not had formal training in mathematical programming. One of the most effective ways of doing this is to present a visual explanation of the problem and the solution at hand. RPM is a network technique which gives a visual identification of the mathematical jargon of mathematical programming.

RPMS was developed by M.S. Inoue and J.L. Riggs at Oregon State University in 1972. It is a concatenation of cause and effect diagrams to model most deterministic type mathematical programming models. The technique has been used to model linear programming, dynamic programming, goal programming and quadratic programming. The most recent significant contribution has been the visual identification of Kuhn-Tucker conditions (Inoue, 1974). This chapter is devoted to discussing the RPM technique.

Consider the following linear programming problem in the canonical form:

$$\text{Max. } Z(x) = \sum_{j=1}^{n} c_j x_j$$

subject to:

$$\text{for } 1 \leq i \leq m$$

(2.1)
\[
\sum_{j=1}^{n} a_{ij} x_j \leq b_i \tag{2.2}
\]

and

\[
x_j \geq 0 \tag{2.3}
\]

The RPMS portrays each of the resource constraints as a cause-and-effect diagram illustrating that the sum of the endogenous \((a_{ij} x_j)\) and exogenous \((b_i^+)\) supply of resource cannot be less than the sum of the endogenous \((a_{ij}^+ x_j)\) and exogenous \((b_i^-)\) demand satisfied by the resource.

The dual of the above primal model can be expressed as

Minimize \(Z(y) = \sum_{i=1}^{m} b_i^+ y_i - \sum_{i=1}^{m} b_i^- y_i\)

subject to

\[
\sum_{i=1}^{m} a_{ij}^+ y_i + c_i^- \geq \sum_{i=1}^{m} a_{ij}^- y_i + c_j^+ \]

for \(1 \leq i \leq m\) and \(1 \leq j \leq n\)

In the primal model the \(n\) constraints are called "process" constraints since they convert endogenous \((\sum_{i=1}^{m} a_{ij}^+ y_i)\) and exogenous \((c_j^-)\) resource flows into the output resource flows \((\sum_{i=1}^{m} a_{ij}^- y_i + c_j^+).\)

As \(y_i\) represents the imputed value of the resource \(i\), and \(c_j^-\) represents the per unit benefit and cost of the transformation, the total value of the input resources and cost of transformation is at least as great as the total value of the output resources and the benefits accrued from this transformation.
By this method all the variables are non-negative, but nothing is said about the sign of the coefficients $a_{ij}$, $b_i$, and $c_j$. These constants can be separated according to their positive and negative components, the components themselves being non-negative. Thus,

$$a_{ij} = a_{ij}^+ - a_{ij}^- \quad (2.4)$$

$$b_j = b_j^+ - b_j^- \quad (2.5)$$

$$c_j = c_j^+ - c_j^- \quad (2.6)$$

1. $a_{ij}^+ = a_{ij}$ if $a_{ij} > 0$ else $a_{ij}^+ = 0 \quad (2.7)$

2. $a_{ij}^- = a_{ij}$ if $a_{ij} < 0$ else $a_{ij}^- = 0 \quad (2.8)$

3. $(a_{ij}^+) \cdot (a_{ij}^-) = 0 \quad (2.9)$

4. $b_i^+ = b_i$ if $b_i > 0$ else $b_i^+ = 0 \quad (2.10)$

5. $b_i^- = b_i$ if $b_i < 0$ else $b_i^- = 0 \quad (2.11)$

6. $(b_i^+) \cdot (b_i^-) = 0 \quad (2.12)$

7. $c_i^+ = c_i$ if $c_i > 0$ else $c_i^+ = 0 \quad (2.13)$

8. $c_i^- = c_i$ if $c_i < 0$ else $c_i^- = 0 \quad (2.14)$

9. $(c_i^+) \cdot (c_i^-) = 0 \quad (2.15)$

Thus the original problem (Eq. 2.1-2.3) can be written as

Maximize \[ Z(x) = \sum_{j=1}^{n} (c_j^+ - c_j^-) x_j \] (2.16)

subject to:
\[
\sum_{j=1}^{n} (a_{ij}^+ - a_{ij}^-) x_j \leq (b_{i}^+ - b_{i}^-) \quad 1 \leq i \leq m
\]  
(2.17)

An expansion of the above gives the primal model

\[
\text{Max } Z(x) = \sum_{j=1}^{n} c_{j}^+ x_j - \sum_{j=1}^{n} c_{j}^- x_j
\]
(2.19)

subject to:

\[
\sum_{j=1}^{n} a_{ij}^- x_j + b_{i}^+ \geq \sum_{j=1}^{n} a_{ij}^+ x_j + b_{i}^- \quad 1 \leq j \leq n
\]
(2.20)

and all variables and parameters are non-negative:

\[
x_j, a_{ij}^+, a_{ij}^-, b_{i}^+, b_{i}^-, c_{j}^+, c_{j}^- \geq 0
\]
(2.21)

for \(1 \leq i \leq m\) and \(1 \leq j \leq n\)

The RPMS portrays each of the resource constraints as a cause-and-effect diagram illustrating that the sum of the endogenous \((a_{ij}^- x_j)\) and exogenous \((b_{i}^+)\) supply of resource cannot be less than the sum of the endogenous \((a_{ij}^+ x_j)\) and exogenous \((b_{i}^-)\) demand satisfied by the resource.

The dual model of the above primal model can be expressed as

\[
\text{Minimize } Z(y) = \sum_{i=1}^{m} b_{i}^+ y_i - \sum_{i=1}^{m} b_{i}^- y_i
\]
(2.22)

subject to:

\[
\sum_{i=1}^{m} a_{ij}^+ y_i + c_{j}^- \geq \sum_{i=1}^{m} a_{ij}^- y_i + c_{j}^+ \quad \text{for } 1 \leq i \leq m \text{ and } 1 \leq j \leq n
\]
(2.23)

In the primal model the \(n\) constraints are called "process" constraints since they convert endogenous \((\sum_{i=1}^{m} a_{ij}^+ y_i)\) and exogenous \((c_{j}^-)\)
resource flows into the output resource flows \( \sum_{i=1}^{m} a_{ij} y_i + c_j^+ \). As \( y_i \) represents the imputed value of the resource \( i \), and \( c_j^+ \) represents the per unit benefit and cost of the transformation, the total value of the input resources and cost of transformation is at least as great as the total value of the output resources and the benefits accrued from this transformation.

**Components of RPMS**

To keep the symbols as simple as possible a basic RPM network is represented by three node symbols and a graphic structure. The three nodal symbols are: 1) resource nodes represented by circles; 2) process nodes represented by squares; and 3) minimizing and maximizing nodes represented by triangles.

A circle is a representation of a limited resource which is taken to mean anything that can place a limitation on the attainment of a desired result. As indicated in Figure 2-la, a circle node is divided into four parts. The top quarter \( y_i \) represents the "shadow price" or the cost of having that resource. Mathematically, it is the value of the Langrange Multiplier (Taha, 1972) associated with the resource constraint.
The bottom quarter $x_i$ represents the residue value $x_i$ of the resource. $y_i$ has the units of "$" or the units of the objective function, whereas $x_i$ has the units of quantity showing the amount of resource left over. This corresponds to the "slack" or "surplus" variable value needed to convert an inequality to an equality.

$$x_i = \sum_{i=1}^{n} a_{ij} x_j + b_i^+ - a_{ij}^+ x_j - b_i^-$$ (2.24)

The other two quarter sections of the circle are optionally used to tally the input flows

$$\sum_{i=1}^{n} a_{ij}^- x_j + b_i^+$$ (2.25)

and the output flows,

$$\sum_{i=1}^{n} a_{ij}^+ x_j + b_i^-$$ (2.26)

through the node.

By complementary slackness theorem either $x_i$ or $y_i$ has to have a zero value. The other two quadrants are used to tally the inflow and outflow.

A square node is used to represent a process where a process is the transformation of available resources to create resources for further uses. This process node represents the primal decision variable in a linear programming model.

The process node is also divided into four parts as shown in Figure 2.1b. The value $x_j$ in the top quadrant is the primal value of
the variable or in other words it represents the amount of the transformation that is to occur. The bottom quadrant $y_j$ represents the "opportunity cost" or the expected loss value of a variable not entering the final basis of a linear programming problem. By the complementary slackness theorem either $x_j$ or $y_j$ has to be zero.

$$y_j = \sum_{i=1}^{m} a_{ij}^+ y_i + c_j^- - \sum_{j=1}^{n} a_{ij}^- y_i - c_j^+$$ \hspace{1cm} (2.27)

The $y_j$ variable can also be interpreted as the Lagrange multiplier associated with the non-negative constraints imposed upon the variable $x_j$. The other two quarter sections of the square may be used to tally the input flows

$$\sum_{i=1}^{m} a_{ij}^+ y_i + c_j^-$$ \hspace{1cm} (2.28)

and the output flows,

$$\sum_{i=1}^{m} a_{ij}^- y_i - c_j^+$$ \hspace{1cm} (2.29)

Contrary to the resource node the top quadrant represents amount and the other three quadrants have the same unit as the objective function.

All internal (endogenous to the system) interrelationships are represented by solid arrows. The value of the arrows represents $a_{ij}$. The value of $a_{ij}$ is non-negative with the arrow direction representing the sign or the flow within a system. The sign of the coefficient $a_{ij}$ is negative when the arrow head is directed away from a circle and
positive when it is directed towards the circle. In summary, the circles, squares, and the solid arrows in the intended directions combined together represent the internal system.

The external or uncontrollable influences on a system are modeled with the help of dashed arrows and triangles. The dashed arrows represent the objective function value if they link squares with a triangle; those that link circles with a triangle represent external constraints on resources. Viewing it from the linear programming equations they represent the $c_j$ and $b_i$ values respectively.

The triangles correspond to the terminal node—one representing the primal objective and the other the dual objective. They are also termed as the "source" or "sink" depending on whether they are connected to the resources or activities.

A maximizing primal objective function is shown by connecting process nodes to the "primal sink" triangle by dashed arrows. The value of the objective function is entered in the triangle (Figure 2.1c).

A minimizing dual objective function is also shown on an RPMS by connecting all the resource nodes and the "source triangle" through dashed arrows with the value of the objective function in the triangle. Reversing a minimizing primal model will have a primal source node and a maximizing dual sink node (Figure 2.1c).
\( n \sum_{j=1}^{n} a_{ij} x_j + b_i \geq \sum_{j=1}^{n} a_{ij}^+ x_j + b_i^- \)

(a) Resource Node

\( m \sum_{i=1}^{m} a_{ij}^+ y_i + c_j^- \geq \sum_{i=1}^{m} a_{ij}^- y_i + c_j^+ \)

(b) Process Node

Maximize \( Z_x = \sum_{j=1}^{n} c_j^+ x_j - \sum_{j=1}^{n} c_j^- x_j \)

Minimize \( Z_y = \sum_{i=1}^{m} b_i^+ y_i - \sum_{i=1}^{m} b_i^- y_i \)

circle = i      square = j

(c) Maximizing and Minimizing

Figure 2-1. RPMS Nodal Conventions.
The internal system connected by solid arrows and the external system linked by dashed arrows represents an RPMS network. Figure 2-2 represents a basic network.

**Postulates of RPMS**

Two characteristics of an RPMS network have been formulated as postulates (Inoue, 1974).

The first postulate addresses the idea of balance around a node within an RPMS network. The second postulate addresses the idea of objective function optimality. These postulates correspond to the revised canonical forms given by equations 2-19, 2-20, 2-22, and 2-23.

**First Postulate of RPMS:** The total inflow at a process or a resource node cannot be smaller than the sum of the outflows from the same node.

**Second Postulate of RPMS:** The productivity of an RPMS network is to be optimized, either by maximizing the net effective endogenous output while holding the exogenous input constant, or by minimizing the exogenous input while maintaining the endogenous output at a given level.
Min $Z_y = b_i y_i$

Max $Z_x = c_j x_j$

$b_i \geq a_{ij} x_j$

$a_{ij} y_i \geq c_j$

$b_i = a_{ij} x_j + x_i$

$a_{ij} y_i = c_j + y_j$

**Figure 2-2.** RPMS Basic Flow

$a_{12} y_1 \geq a_{23} y_3 + c_2$ or $a_{12} y_1 = a_{23} y_3 + c_2 + y_2$

$a_{12} x_1 \geq a_{23} x_3 + b_2$ or $a_{12} x_1 = a_{23} x_3 + b_2 + x_2$

**Figure 2-3.** RPMS Cause and Effect Diagrams.
RPMS Conventions

In addition to the two postulates of RPMS, several conventions exist for the construction of RPMS networks (Riggs and Inoue, 1975).

The main conventions are presented here as a convenience to the reader. These conventions are used throughout this study.

1. A circle never connects to a circle and a square never connects to a square directly.

2. All squares are explicitly or implicitly connected to one terminal, and all circles are explicitly or implicitly connected to the other terminal.

3. The dimension of the solid arrow coefficient is always (Resource Unit/Process Unit).

4. Each circle adheres to the logic rules for "inclusive or."
5. Each square adheres to the logic rules for "logical product."

Construction of the RPMS Network

The actual construction of an RPMS network is relatively simple once the linear programming problem has been formulated. The construction consists of the following steps.

1. Draw a square for each variable in the objective function.
2. Identify the optimization type required in the objective function (maximize or minimize) and add the appropriate terminal nodes to the existing diagram (use Postulate Two).
3. Draw a circle for each constraint.
4. Complete the network using the equations and Postulate One.

The RPMS network is now ready to accept additional information after the linear programming problem has been solved.

Advantages of RPMS Networks

The biggest advantage to this form of network analysis is that it is an excellent communication tool. It bridges the gap between the quantitative analyst, the shop floor supervisor and the manager at various levels of decision making. This is one of the objectives of this thesis. It is also a very efficient method of performing post-optimality analysis to indicate places where further improvements can
be made for the benefit of the total system. Figure 2-5 shows the conditions which can exist in RPMS networks. The ideas in this figure were obtained from the paper on Visual Identification of Kuhn-Tucker conditions (Inoue, 1974).

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
<th>RPMS Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Non-optimal</td>
<td>One or more negative $y_j$ values</td>
<td><img src="insert%20image" alt="negative yj" /></td>
</tr>
<tr>
<td>B Non-feasible</td>
<td>One or more negative $x_i$ values</td>
<td><img src="insert%20image" alt="negative xi" /></td>
</tr>
<tr>
<td>C Feasible and optimal</td>
<td>No negative values for $x_i$ or $y_j$</td>
<td><img src="insert%20image" alt="non-negative x and y" /></td>
</tr>
<tr>
<td>D Degeneracy</td>
<td>An alternative solution exists for the same objective function value</td>
<td><img src="insert%20image" alt="degeneracy" /></td>
</tr>
</tbody>
</table>

Figure 2-5. RPMS feasibility and optimality conditions.

RPM networks have a very direct effect on "LPRPM" since both the input and output are made to conform to RPM. Consider the problem:

$$\text{Max } Z(x) = 2x_1 + x_2$$

subject to:
\[3 x_1 + x_2 = 3\]
\[4 x_1 + 3x_2 \geq 6\]
\[x_1 + 2x_2 \leq 3\]
\[x_1, x_2 \geq 0\]

(2.30)

This problem cannot be entered directly into LPRPM, since the routine is designed to accept only "less than" type constraint equations. However, a RPM representation of the above problem would very easily take care of everything. RPM network of 3 would be

In accordance with the RPMS conventions this is a feasible but non-optimal solution to the problem (2.30).

The network may be expressed in equation form as
Max \( Z(x) = 2x_1 + x_2 \)

subject to:

\[
\begin{align*}
3x_1 + x_2 & \leq 3 \\
-3x_1 - x_2 & \leq -3 \\
-4x_1 - 3x_2 & \leq -6 \\
x_1 + 2x_2 & \leq 3 \\
x_1, x_2 & \geq 0
\end{align*}
\]

As far as using LPRPM is concerned, the numbers can be directly entered from the network and the results to the network.
III. A REVIEW OF THE GENERAL LINEAR PROGRAMMING ALGORITHMS

In this chapter the general forms of linear programming problems and some of the algorithms used for their solution will be reviewed. The intention is not tutorial but rather to summarize the state of the art and also to discuss their merits and demerits in terms of applicability to linear programming computer codes.

Definition of Linear Programming

Mathematically, linear programming is defined as the problem of finding a maximum (or minimum) of a linear function, subject to linear non-negativity constraints (Dantzig, 1963). The term linear programming was coined in 1947 to describe "the process of planning a program on the basis of linear constraints of the level of activities" (Beale, 1968); it does not necessarily mean writing computer programs to solve linear problems although computers are always used to handle large linear programming problems.

General Forms of Linear Programming Problems

Canonical Form

There are two common forms to represent a linear programming model: the canonical form and the standard form. A general linear
programming problem can be defined in the canonical form as

\[
\text{Max } Z(x) = \sum_{j=1}^{n} c_j x_j
\]  

(subject to the constraints:

\[
\sum_{j} a_{ij} x_j \leq b_i \quad i = 1, 2, \ldots, n \tag{3.2}
\]

\[
\sum_{j} x_j \leq 0 \quad j = 1, 2, \ldots, n \tag{3.3}
\]

The characteristics of this form are:

1. All decision variables are non-negative.
2. All constraints are of the \( \leq \) type.
3. The objective function is maximizing.

All LP problems can be converted to a canonical form by the use of five elementary transformations:

1. Minimization of functions is equivalent to maximization of the negative of the function, \(-f(x)\). For example:

\[
\text{minimize } Z(x) = c_1 x_1 + c_2 x_2 + c_3 x_3 + \cdots + c_n x_n \tag{3.4a}
\]

is equivalent to:

\[
\text{maximize } Z(x) = -c_1 x_1 - c_2 x_2 - c_3 x_3 - \cdots - c_n x_n \tag{3.4b}
\]

Consequently, all LP problems can be put into the maximization form.

2. An inequality in one direction (\( \leq \) or \( \geq \)) may be changed to an inequality in the opposite direction by multiplying by \(-1\), such as:

\[
a_1 x_1 + a_2 x_2 \geq b_1 \tag{3.4c}
\]
3. An equality may be replaced by two inequalities in opposite directions, such as:

\[ a_1 x_1 + a_2 x_2 = b_1 \]  

is equivalent to:

\[ a_1 x_1 + a_2 x_2 \leq b_1 \text{ and } a_1 x_1 + a_2 x_2 \geq b_1 \]

(3.4f)

4. An equality constraint with left hand side in absolute form can be changed to two regular inequalities. Thus, for \( b_1 \leq 0 \):

\[ a_1 x_1 + a_2 x_2 = b_1 \]

is equivalent to:

\[ a_1 x_1 + a_2 x_2 \leq -b_1 \text{ and } a_1 x_1 + a_2 x_2 \geq b_1 \]

(3.4g)

(3.4h)

5. A variable which is unconstrained in sign is equivalent to the difference of two non-negative variables. Thus, if \( x \) is an unconstrained variable it can be replaced by \( (x^+ - x^-) \) where \( x^+ \) and \( x^- \) are both non-negative, or \( x^+ \geq 0 \text{ and } x^- \geq 0 \) (Taha, 1971)

**Standard Form**

The characteristics of the standard form are:

1. All the constraints are expressed as equality type equations except for the non-negativity constraints which remain of the type \( x^+ \geq 0 \).
2. The right hand side elements of the constraints have to be non-negative.
3. All variables are to be non-negative.
4. The objective function could be either maximization or minimization type.

An inequality constraint can be easily changed to an equality by adding a "slack variable" in the case of a \( \leq \) constraint or by subtracting a "surplus variable" in the case of \( \geq \) constraint. A negative right hand side of an equation may always be made positive by multiplying the equation by (-1) (Hadley, 1963; Wagner, 1969; Taha, 1971).

**Concept of a Basis**

A candidate for the starting solution for a linear programming problem may be selected by setting any \( n \) primal variables (out of \( m+n \) primal variables) to zero and then solving for the remaining \( m \) unknowns, provided that the solution exists and is unique. In this case the \( n \) zero variables are called "nonbasic" variables; the remaining \( m \) variables are called "basic" variables, and constitute a "basic" primal solution (Taha, 1971).

Consider the set of \( m \) independent linear equations in \( n \) unknowns (\( m \leq n \)):

\[
AX = b
\]  

(3.5)

where \( X \) represents a vector of \( n \) unknowns. Then the conditions of the aforementioned discussion can be represented as :

\[
A = (P_1, P_2, \ldots, P_n)
\]  

(3.6)

where \( P_j, j = 1, 2, \ldots, n \) represents the \( j \)th column of \( A \) vector.
Then the basic solution is obtained by setting the variables corresponding to \((n-m)\) vectors equal to zero and solving for the remaining variables, provided the square matrix comprised of these vectors is nonsingular.

**A Survey of the Algorithms**

The first viable method presented for solving linear programming problems was the simplex algorithms developed by G. Dantzig. This method has been widely documented and serves as a backbone of most linear programming studies.

Before delving into the issues raised by this method, we should consider the geometrical interpretation of a linear programming problem. Any problem with \(n\) nonbasic variables can be represented in an \(n\)-dimensional diagram (Beale, 1968, p. 25). Figure 3.1 represents a problem with two independent variables \((x_1\) and \(x_2)\) with lines corresponding to the four constraints. The arrow indicates the direction of travel of the linear objective function. Mathematically the problem can be represented as:

\[
\text{Max } Z(x) \leq \sum_{j=1}^{n} c_j x_j \quad (3.7a)
\]

where \(n = 2\)

subject to:
\[
\sum_{j} a_{ij} x_j \leq b_i \quad (3.7b)
\]

where \(m = 4\)

and
\[
\sum_{j} x_j \geq 0 \quad (3.7c)
\]
Referring to Figure 3.1 the feasible region is represented by a polygon; or, in a more general case, a polyhedron with as many dimensions as the number of nonbasic variables.

**The Simplex Procedure**

The simplex method is an algebraic iterative procedure which will solve exactly any linear programming problem in a finite number of steps, or give an indication that it is an unbounded solution (Hadley, 1963). It is a procedure for moving a given extreme point to an adjacent extreme point along an "edge" of the "feasible region." The process stops once the optimal extreme point is reached. The optimal extreme point is the point at which the greatest value (or the least value in case of a minimization problem) of the objective function is obtained.

The simplex algorithm starts with a nonoptimal, feasible solution and proceeds through a finite number of iterations to achieve an
optimal solution. It is essential to make sure that feasibility conditions are satisfied at each step.

Variations of the Simplex Algorithm

There are several variations of the simplex algorithm, one of which is the two-phase method (Taha, 1971). In this, Phase I minimizes the objective function which is comprised of the sum of the artificial variables only. The minimum value of the objective function will be zero if a feasible solution exists.

The optimal solution is obtained in Phase II by using the basic feasible solution from Phase I as a starting solution for the original problem.

The dual-simplex algorithm is an application of the simplex algorithm pivot selection rules to the dual problem (Lemke, 1954). The dual-simplex procedure starts with an optimal, nonfeasible solution, signified by the right-hand side being negative (Taha, 1971).

The primal-dual algorithm is used to solve the primal and dual problems simultaneously (Dantzig et al., 1956). The algorithm alternates between the primal and dual constraint systems, optimizing the decision variables with respect to the complementary slackness theorem.

These algorithms, besides providing proven results for linear programming problems, also possess the advantage of conceptual
simplicity. However, the additional computational and setup work and the need for excessive amounts of computer memory space make it necessary to develop new solution techniques. The most prominent ones include the revised simplex method, the decomposition algorithm, and the product form of the inverse.

**Matrix Form of LP**

The general LP problem can be defined in the matrix form as:

\[
\text{Max } Z(x) = CX
\]  

subject to:

\[
(A, I)X = P_o \tag{3.9}
\]

\[
X \geq 0 \tag{3.10}
\]

where \( C \) represents the vector of the coefficients of the objective function, \((A, I)\) represents the current nonbasic vectors, \( B \) is the basic matrix and \( X_B \) the basic variables. At any iteration the current basic solution is given by the equation:

\[
BX_B = P_o \tag{3.11}
\]

and

\[
Z(x) = C_B X_B \tag{3.12}
\]

where \( C_B \) represents the vector coefficients of the variables in the basis.

Combining equations 3.11 and 3.12 we have:

\[
\begin{bmatrix}
1 - C_B \\
- B
\end{bmatrix}
\begin{bmatrix}
Z(x) \\
X_B
\end{bmatrix} = 
\begin{bmatrix}
0 \\
P_o
\end{bmatrix} \tag{3.13}
\]
where:

\[
M = \begin{bmatrix}
1 & -CB \\
B & B
\end{bmatrix}
\]  
(3.14)

Then equation 3.13 can be written as:

\[
M \begin{bmatrix} Z(x) \\ X_B \end{bmatrix} = \begin{bmatrix} 0 \\ P_o \end{bmatrix}
\]  
(3.15)

Thus the current solution is given by:

\[
\begin{bmatrix} Z(x) \\ X_B \end{bmatrix} = M^{-1} \begin{bmatrix} 0 \\ P_o \end{bmatrix}
\]  
(3.16)

where:

\[
M^{-1} = \begin{bmatrix}
1 & CB \\
0 & B^{-1}
\end{bmatrix}
\]  
(3.17)

The matrix definition of the problem and its solution has a direct effect on the storage space requirement because one is ultimately dealing with basic matrix B only rather than the entire tableau.

The development of the revised simplex method produces savings in the computational time. This new technique allows the calculations of the \( B^{-1}_{\text{next}} \) directly from the \( B^{-1}_{\text{current}} \) without going back to the raw data. This is done by use of an E vector, such that:

\[
B^{-1}_{\text{next}} = EB^{-1}_{\text{current}}
\]

where the E vector is formed from the element of the entering vector rather than the original elements.

It should be noted that \( B^{-1} \) for the first iteration is always an identity matrix so through all iterations of this revised simplex method.
method, it will not be necessary to invert any matrices.

The decomposition algorithm is applied where the structure of a large LP problem can be decomposed into smaller subproblems and then solving them independently makes them computationally feasible. This means that the problem will have two types of resources—common constraints and independent constraints (Dantzig, 1963).

**Product Form of Inverse**

The product form of inverse has been the most popular method for building large computerized LP routines (Beale, 1968). *REXY of the CDC 3300 (Lynn Scheurmann, OSU) and MPSX of IBM 370 (IBM, Software Manual) use this method. This method of solution is also used in the proposed software package.

Most industrial problems have a large number of zero values for the coefficients \( a_{ij} \) and \( c_j \). This method saves space by storing only the portion of the matrix that is essential and discarding the useless zeroes and ones that can be recreated from other available information.

The matrix \( B \) in equation 3.11 is sometimes called the elementary column matrix. Its characteristics differ from the identity matrix in only one column. The inverse of this matrix is needed to calculate \( M^{-1} \) of 3.17. The inverse of an elementary column
matrix can be readily computed by taking certain ratios of the pivot elements.

The matrix $B^{-1}$ is also an elementary column matrix. The new update matrix is merely a product of the original matrix and the elementary column matrix $B^{-1}$.

$$\begin{align*}
(A, I)_{\text{new}} &= B^{-1}(A, I)_{\text{current}} \\
\end{align*}$$

(3.18)

Hence this method is named product form of the inverse.

LPRPM utilizes the product form of the inverse method owing to its high efficiency in storage requirements. A numerical example of this procedure is presented in Chapter IV.

**Thompson's Method**

In the discussions presented in the earlier portion of this chapter it was seen that the simplex algorithm depended on the assumption that the right hand side ($b$ vector) was non-negative. Because of this, the "Big M" method and the two-phase methods were introduced. A third, little known method was suggested by G. L. Thompson (1973). We shall now discuss the method and its computational advantages over other methods.

Consider a standard LP problem in the canonical form:

$$\begin{align*}
\text{Max } Z(x) &= \sum_{j} c_{j} x_{j} \\
\end{align*}$$

(3.19)
subject to:

\[ \sum_{j} a_{ij} x_j \leq b_i \]  \hspace{1cm} (3.20)

where

\[ 1 \leq i \leq m \hspace{0.5cm} 1 \leq j \leq n \]

\[ x_j \geq 0 \]  \hspace{1cm} (3.21)

This same problem may be expressed as:

\[ \text{Max } Z(x) = \sum_{j} c_{j} x_j + kx_{n+1} \]  \hspace{1cm} (3.22)

subject to:

\[ \sum_{j} a_{ij} x_j - b_i x_{n+1} \leq 0 \]  \hspace{1cm} (3.23)

\[ x_{n+1} \leq 1 \]  \hspace{1cm} (3.24)

\[ x_j, x_{n+1} \geq 0 \]  \hspace{1cm} (3.25)

Equations 3.22 to 3.25 express the same problem as that represented by 3.19 to 3.21 if \( x_{n+1} \) is set equal to 1. To assume that \( x_{n+1} \) is equal to 1 two conditions are imposed:

1. \( k \) is a large number which makes \( x_{n+1} \) the most probable vector to enter the basis at the maximum primal level, and

2. equation 4.2c ensures that \( x_{n+1} \) enters at a maximum value of 1.

This method has two advantages:

1. It is computationally less cumbersome and easier to handle than the "Big M" method. Only one large number location is necessary; this is for the dual value of \( x_{n+1} \). The objective
function value is obtained by taking the difference between $Z(x)$ and $k$.

2. It always assures a starting basic feasible solution.

Thompson's method was used to set up the problem in LPRPM. The mechanics of converting a standard problem to the form of Thompson's method is handled internally, or built into the proposed computer package.
IV. LPRPM ALGORITHM

The method used in LPRPM can be divided into the following categories.

1. Data entry from an RPM network.
2. Problem setup using Thompson's method.
3. Computation using the product form of the inverse method.
4. Output in the form ready for entry to RPM network.

The first, second and fourth steps have been discussed in Chapters II and III so in this chapter the computational method along with a numerical example is described. This method is sometimes called the condensed tableau method.

**Condensed Tableau Method**

Consider the linear programming problem

\[ \text{Maximize } Z(x) = c_1 x_1 + c_2 x_2 + \ldots + c_j x_j \quad (4.1) \]

subject to:

\[ a_{11} x_1 + a_{12} x_2 + \ldots + a_{1j} x_j \leq b_1 \]
\[ a_{21} x_1 + a_{22} x_2 + \ldots + a_{2j} x_j \leq b_2 \quad (4.2) \]
\[ \vdots \]
\[ a_{i1} x_1 + a_{i2} x_2 + \ldots + a_{ij} x_j \leq b_i \]

\[ x_1, x_2, \ldots, x_j \geq 0 \quad (4.3) \]
By applying Thompson's method the same problem can be expressed as

Maximize

\[ Z(x) = c_1 x_1 + c_2 x_2 + \ldots + c_j x_j + k x_{j+1} \]

subject to:

\[
\begin{align*}
y_1 &= a_{11} x_1 + a_{12} x_2 + \ldots + a_{1j} x_j - b_1 x_{j+1} \leq 0 \\
y_2 &= a_{21} x_1 + a_{22} x_2 + \ldots + a_{2j} x_j - b_2 x_{j+1} \leq 0 \\
&\vdots \\
y_i &= a_{i1} x_1 + a_{i2} x_2 + \ldots + a_{ij} x_j - b_i x_{j+1} \leq 0 \\
y_{i+1} &= x_{j+1} \leq 1
\end{align*}
\](4.4)

\[ x_1, x_2, \ldots, x_j, x_{j+1} \geq 0 \]

The matrix representation of the above problem is as follows.

(m + 1 x n + 2) matrix

\[
\begin{array}{cccc|c}
 x_1 & x_2 & x_j & x_{j+1} & \text{Obj} \\
 \hline
 c_1 & c_2 & \ldots & c_j & k \\
 a_{11} & a_{12} & \ldots & a_{ij} & -b_1 \\
 \vdots & \vdots & \ddots & \vdots & \vdots \\
 a_{i1} & a_{i2} & \ldots & a_{ij} & -b_2 \\
 0 & 0 & \ldots & 0 & 1 \\
 \hline
 & & & & 1 \\
 \end{array}
\]
The tableau has the primal values along the top row and the dual values along the last column. As compared to simplex method this is very highly compact and no pertinent information is lost. The steps to solution process are as follows.

1. Test for feasibility by checking that the \(n+2\) column (\(b_i\) vector) has all non-negative values. Then test for optimality by checking the first row (\(c_j\) vector) for negative value. If all values in the first row are positive an optimal solution has been reached.

2. Select a pivot column by determining the most negative value in the first row (\(c_j\) vector).

3. Select a pivot row by determining the minimum of the ratio of each element in \(n+2\) column (\(b_i\) vector) and the corresponding positive nonzero element in the pivot column. The intersection of this pivot column and the pivot row gives the pivot element.

4. Construct the pivot vector by taking the negative ratio of the elements in the pivot column and the pivot element, except for the pivot element position. The pivot element position becomes 1.

5. Determine the solution matrix by multiplying the pivot matrix by each column of the current matrix. This is the product of \([ (m+1) \times (m+1) ]\) and \([ (m+1) \times 1 ]\) matrices.
It is important to note that the pivot matrix \((m+1 \times m+1)\) is an identity matrix except for the pivot column itself. So it is necessary to carry only the pivot column in the core. This particular point is very important in saving storage spaces.

5. Go back to 2 for next iteration.

**Numerical Example**

Consider the problem:

\[
\text{Max } Z(x) = x_1 + 2 \times 2
\]

subject to:

\[
\begin{align*}
2 \times x_1 + x_2 & \leq 8 \\
-x_1 + x_2 & \leq 4 \\
x_1, x_2 & \geq 0
\end{align*}
\]

\[(4.6)\]

The problem is expressed in the form suggested by G. L. Thompson by adding a variable \(x_3\)

\[
\text{Max } Z(x) = x_1 + 2 \times 2 + 1000 \times 3
\]

subject to:

\[
\begin{align*}
2 \times x_1 + x_2 - 8 \times x_3 & \leq 0 \\
-x_1 + x_2 - 4 \times x_3 & \leq 0 \\
x_3 & \leq 1 \\
x_1, x_2, x_3 & \geq 0
\end{align*}
\]

\[(4.7)\]
To solve the problem it is necessary to set it up in condensed tableau form.

\[
\begin{array}{cccc|c}
\hline
\ x_1 \ & \ x_2 \ & \ x_3 \ & \text{Obj} \\
\hline
-1 & -2 & -1000 & 0 \\
2 & 1 & -8 & 0 \\
1 & 1 & -6 & 0 \\
-1 & 1 & -4 & 0 \\
0 & 0 & 1 & 1 \\
\hline
\end{array}
\]

\[\begin{bmatrix}
1000 \\
8 \\
6 \\
4 \\
1 \\
\end{bmatrix}\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1000 \\
0 & 1 & 0 & 0 & 8 \\
0 & 0 & 1 & 0 & 6 \\
0 & 0 & 0 & 1 & 4 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

The last two steps are internally done by LPRPM.

Pivot column = 3

Pivot row = 5

Pivot element = 1

Iteration 1: The new matrix can be obtained by multiplying the pivot matrix and each column of the current matrix. It is important to note that the pivot matrix is an identity matrix with the exception of the
the pivot column so it is necessary to carry only the pivot column in this matrix.

**Iteration 1:**

The new matrix is

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_4)</th>
<th>1000</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1)</td>
<td>(-2)</td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>(1)</td>
<td>(8)</td>
<td>(8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>(1)</td>
<td>(6)</td>
<td>(6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1)</td>
<td>(1)</td>
<td>(4)</td>
<td>(4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0)</td>
<td>(0)</td>
<td>(1)</td>
<td>(1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Value of objective function = 1000 - 1000 = 0.
Solution is feasible but non-optional.

Pivot column = 2

Pivot row = 4

Pivot element = 1

Pivot vector

\[
\begin{bmatrix}
2 \\
-1 \\
-1 \\
0
\end{bmatrix}
\]

**Iteration 2:**

The new matrix is

<table>
<thead>
<tr>
<th></th>
<th>(x_1)</th>
<th>(y_3)</th>
<th>(y_4)</th>
<th>1008</th>
<th>1008</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-3)</td>
<td>(2)</td>
<td>1008</td>
<td>1008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td>(-1)</td>
<td>(4)</td>
<td>(4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>(-1)</td>
<td>(2)</td>
<td>(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-1)</td>
<td>(1)</td>
<td>(4)</td>
<td>(4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0)</td>
<td>(0)</td>
<td>(1)</td>
<td>(1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Value of objective function = 1008 - 1008 = 0.
Solution is feasible but non-optional.
Value of objective function = 8
Solution is feasible but non-optimal.
Pivot column = 1

Pivot row = 3
Pivot element = 2

Iteration 3:

Pivot vector
\[
\begin{bmatrix}
3/2 \\
-3/2 \\
1/2 \\
1/2 \\
0
\end{bmatrix}
\]

The new matrix is
\[
\begin{array}{ccccc|c}
& y_2 & y_3 & y_4 & \text{Obj} \\
3/2 & 1/2 & 1011 & 1011 & y_1 \\
-3/2 & 1/2 & 1 & 1 & x_1 \\
1/2 & -1/2 & 1 & 1 & x_2 \\
1/2 & 1/2 & 5 & 5 & x_3 \\
0 & 0 & 1 & 1 & \\
\end{array}
\]

Value of objective function = 11

Optimal solution is reached since there is no -ve indicators in Row 1.
V. CORE SPACE ALLOCATION ON WANG 2200

Storage efficiency was the main concern while developing this routine. On the Wang 2200 minicomputer that was used, the maximum amount of information that can reside in the core at any time is limited to 4000 bytes. This problem of lack of storage space is handled in two ways:

1. Minimizing program storage space requirements.
2. Saving only the essential (non-zero) elements in core.

All of the remaining information is stored in an auxiliary storage device such as a cassette tape in this case.

Program Storage

To save program storage space requirements, the entire routine is divided into eight subprograms. These are almost like subroutines in a Fortran program, with the exception that only one subprogram can reside in the core at any given time. Wang 2200 permits the user to clear the memory, load a new program, and save the values associated with the "common" designated variables by the use of a single programmable statement. All the other sections are stored on cassette.

Wang 2200 also allows packing of more than one BASIC statement in one line. One hundred thirty-eight characters, including
blanks, can be packed in one line. Each line of information is stored in memory with a "beginning of record" (BOR) and an "end of record" (EOR). By packing more than one statement in a line fewer "BOR" and "EOR" are necessary; consequently, storage space requirements are lessened.

**Data Storage**

To achieve maximum efficiency in data storage space requirements and yet not lose accuracy, several different data manipulation methods were tried. Just as in Standard BASIC language two different types of variables are permitted: numeric and the string variable. String variables can store any character; however, for computational purposes, all numeric data must be represented by numeric variables. Owing to the convenience in handling and manipulation, string variables were used to store all the data and are only converted to numeric variables before doing any computations.

On Wang 2200, PACK statements are used to pack numeric data into string variables using one byte for two digits. The ":" sign takes up one-half of a byte, and the exponent including its sign takes up one byte. An error results if the string variable is not large enough to store all of the numeric values to be packed. For this, it is necessary to dimension the string variables to the required length at the beginning of the program.
In order to be able to solve a problem with as many as 100 constraints within the space limitations, it was necessary to dimension the string variables to use four bytes per variable, which allowed the storage of five to seven digits depending on the form of the numbers to be stored. This then brought about the problem of round off errors.

Round off errors was one of the most important problems faced in the proposed routine. A short example illustrates the magnitude of the round off error buildup. Consider the sum of three numbers: 2.5, 3.5, and 4.5:

\[
\begin{align*}
2.5 + 3.5 + 4.5 &= 10.5 \quad (5.1) \\
3 + 4 + 5 &= 12 \quad (5.2) \\
2 + 3 + 4 &= 9 \quad (5.3) \\
3 + 3 + 4 &= 10 \quad (5.4)
\end{align*}
\]

Equation 5.1 represents the correct answer. If all the numbers were rounded off then the result in equation 5.2 would be obtained; the result 5.3 is obtained by truncating and not by rounding at all. Equation 5.4 represents the result of randomly rounding and truncating; it is obvious that equation 5.4 yields the closest result to the correct answer. In the LPRPM routine, every time a decimal number ending with 5 has to be rounded, a random number is generated and the rounding is done randomly.
The crux of the LPRPM routine depends on the auxiliary device, i.e., cassette tapes. All the data and other information are always stored on cassettes; only the pieces of information that are necessary for immediate calculations are brought into core and are returned to the cassette immediately after the calculations. This process of packing and unpacking of data in turn builds up round off errors. To minimize round off errors, two methods were tried: the logarithm and the exponent form.

**Logarithm Form**

The idea of this method is to store all numbers in their logarithmic form and to do all multiplications and divisions by simply adding and subtracting the logarithms. However, since the sums and differences of numbers cannot be done in the logarithm form, it entailed a conversion to real numbers, and thus added round off errors. Also, since the logarithm of a negative number does not exist, both the signs of the real number and the sign of the logarithm of the absolute value had to be stored. If the real number is negative less than 1 then its logarithm value is negative; to avoid the confusion between the signs, after taking the logarithm of a number, 5 was added to it to make it positive and then it was stored along with the sign of the real number. The reverse procedure was applied to convert to real numbers.
Example:

To store the nos. \( x = 55.95 \) and \( y = -.0042 \) the following steps were followed:

1. Natural logarithm of the numbers was determined.

\[
A = \ln(55.95) = 4.024458435 \\
B = \ln(.0042) = -5.472670754
\]

2. Add 5.00 to \( A \) and \( B \) to make sure they are positive.

\[
A = 9.024458435 \\
B = 0.472670754
\]

3. Check the signs of \( x \) and \( y \) and assign the same to \( A \) and \( B \).

\[
A = 9.024458435 \\
B = -0.472670754
\]

4. Correct \( A \) and \( B \) to six decimal places.

\[
A = 4.024458 \\
B = -5.472671
\]

Pack \( A \) and \( B \) into four bytes of memory space.

Note if the seventh digit of any number is "5" then while correcting, the following special steps are followed:

1. Pick a random number and multiply it by 2.

2. If the resulting random number is greater than 1, then correct the original number "up" or else do not correct "up."

The steps 1 to 4 were applied in reverse order to obtain the real number. Each time a conversion was performed a certain magnitude of error was introduced. To reduce this error buildup, all multiplying and divisions were done in the logarithm form without
converting; however, for addition and subtraction the numbers had to be converted.

This method was very effective for storing numbers which were less than $10^{-2}$ and also those that were larger than $10^2$. However, the conversion of logarithms to real numbers during the process of addition and subtraction introduced considerable magnitude of error; therefore, this method was rejected.

**Exponent Form**

In the exponent form the data are stored in their exponential form. By packing the data and their exponent into four bytes of memory, it is possible to save five non-zero numbers. Also, the round off errors using this method are considerably lower. This method was finally adopted.

A comparison of the results obtained by LPRPM and other accepted routines are presented in the Appendix.

**Pointer System**

Along with the method of packing the data in the string variable, it was also necessary to ensure that only the non-zero data were being stored. To do this it was necessary to devise a pointer system. This is done by storing the binary value of $i$ and $j$ in two bytes of memory along with the value of the coefficient packed in the rest of the
memory. The whole combination of numbers represents the exact location and the value of one non-zero coefficient. As long as the values of i and j are less than 254, there is no problem because the binary is a single digit.

Example:
To store b(15, 102) = 75.35, the following steps were followed:

1. Find the binary of 15 by using the BIN statement and store it in the first byte of the string variable "X$" by using the POS statement.

2. Find the binary of 102 and store it in the second byte of "X$" by using the BIN and POS statements.

3. PACK the number 75.35 in its exponent form into "X$". To convert back to the original number use the VAL instead of BIN statement.
VI. EVALUATIONS AND RECOMMENDATIONS

It was the purpose of this thesis to 1) design an algorithm efficient enough to solve linear programming problems with up to 250 variables and 100 constraints, using a minicomputer with small core memory space (4000 bytes), and 2) design an input-output format which is compatible with the RPMS notation.

Computational Performance

The approach proposed in this thesis has been found to be successful in both of these areas. A comparison study was made by using two other linear programming packages, *REXY and RPMI, for solving four benchmark programming problems. The Linear Programming packages that were used for the comparison are as follows:

1. *REXY - Linear Programming package available on CDC 3300 as prepared by Lynn Scheurman (Scheurman 1970).
2. RPMI - Linear Programming package available on CYBER-73 version of CDC 6400 as prepared by Steve Chou (Chou 1975).
3. LPRPM - Linear Programming package developed for use on Wang 2200 minicomputer and written in BASIC.

The following comparative results were identified in favor of the LPRPM package:

1. CPU Memory Space Requirement - Since the word sizes are
different for the three computers the number of characters was used as a common base for comparison. The problem that was used for comparison had 11 (67) variables and 11 (57) constraints. The approximate CPU memory space requirements were as follows:

*REXY: 68480 (11280) characters
RPMI: 317760 (327920) characters
LPRPM: 2576 (3312) characters

2. Cost of Running - *REXY: $1.02 (1.98)
   RPMI: $0.508 (0.998)
   LPRPM: $0.3

The cost for LPRPM was computed from the cost/min, based on $5000.00 initial cost and 5 year payback period.

Built-in Debugging Aids

An infeasible problem was formulated with 15 variables and 16 constraints and was run using all the three routines, and the availability of information, to debug the infeasibility in the problem, was studied.

*REXY gave very minimal information as to the cause of infeasibility and the iteration no where it was detected. On further investigation it was found that it is possible to obtain the internal calculations performed by the computer. However, this information
was completely mathematical which would be of little use without prior mathematical background.

RPMI incorporates a feature which allows the user to force feasibility by executing a single command. This then gives a good indication of the location of the problem in the model.

LPRPM always starts with a feasible solution so the first iteration is always performed. In case a problem is infeasible the user has the choice to access the current tableau. The output follows a format that is compatible with the RPM network and by transferring the numbers to the network the user gets a clear indication of the problem with his model.

**Areas of Improvements**

The major difficulties faced with the proposed method are:

1. **Errors** - Economizing storage space has led to a buildup of round-off errors as the iterations proceed. The rate of error buildup is approximately .000005 in eight iterations. Each storage location has to be reduced in size to store numbers with up to five decimal places and to increase this accuracy it would mean the need for more storage space. In addition frequent tape read errors were encountered causing system interrupt.

2. **Solution time requirement** - It was seen that with the LPRPM a problem of size 11 variables and 11 constraints took approximately 4 min. per iteration. However, about 10 sec. of that
was used for computation and the rest for data transfer and the mechanical movement of the cassette.

With the contemplated addition of a dual port disk drive as auxiliary memory, the time per iteration is expected to be reduced by 90%. The use of disk drive would also eliminate the problem of tape read error. The tape read error is possibly caused by a change in the length of the tape which, in turn, could be due to various reasons such as change in temperature, stretching of tape due to continuous back and forth motion, etc. The problem of round-off errors cannot be remedied unless the word size is increased to accommodate more significant digits.

**Recommendations for Future Studies**

The principles used in this thesis for data storage (Chapter V) are certainly not new ideas. However, their application to minicomputers for solving mathematical programming problems is new. Using these principles, similar packages can be developed for integer programming, goal programming, PERT, CPM, and inventory control models. The trend towards mathematical analysis of present day management problems coupled with the increased use of minicomputers warrants the development of more powerful mathematical programming software packages. Increased computational speeds to facilitate management use of linear programming and automation of
RPM network drawing using the computer data files would be a useful contribution. At the present time the network has to be drawn by hand and the values are entered manually. It would be very useful to have a completely automated package with a secondary routine to transfer the values for LPRPM to the RPM network.

The research described in this thesis proved that limited CPU memory size is not necessarily deterrent to the size of the linear programming problem that can be solved by a minicomputer. Increasing the computational accuracy and decreasing the overall time requirement to obtain meaningful solutions to management problems will lead to a wide acceptance of Operations Research techniques by the industry.
BIBLIOGRAPHY


APPENDIX 1

LPRPM EXAMPLES
Problem No. 1

Max $Z_x = x_1 + 2x_2$

Subject to

$2x_1 + x_2 \leq 8$
$x_1 + x_2 \leq 6$
$-x_1 + x_2 \leq 5$
$x_1, x_2, x_3 \geq 0$

SOLUTION IS OPTIMAL

NO. OF ITERATIONS = 3

VALUE OF OBJECTIVE FUNCTION = 11

RESOURCES (ROWS) (CIRCLES)

<table>
<thead>
<tr>
<th></th>
<th>P. RESIDUE (BOT. QUADRANT)</th>
<th>D. RESIDUE (TOP QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YTWO</td>
<td>0.000</td>
<td>1.500</td>
</tr>
<tr>
<td>YTRE</td>
<td>0.000</td>
<td>0.500</td>
</tr>
<tr>
<td>YONE</td>
<td>1.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

PROCESSES (COLUMNS) (SQUARES)

<table>
<thead>
<tr>
<th></th>
<th>P. VALUE (TOP QUADRANT)</th>
<th>D. VALUE (BOT. QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XONE</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>XTWO</td>
<td>5.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Problem No. 2

Max \( Z_x = 4x_1 + x_2 \)

Subject to

\[
\begin{align*}
  x_1 & \leq 30 \\
  -x_1 & \leq -30 \\
  x_1 & \leq 50 \\
  6x_1 + 6x_2 & \leq 360 \\
  x_2 & \leq 40 \\
  -x_2 & \leq -20 \\
  x_1, x_2 & \geq 0
\end{align*}
\]

Solution is Optimal

No. of Iterations = 5

VALUE OF OBJECTIVE FUNCTION = 149.7

<table>
<thead>
<tr>
<th>RESOURCES (ROWS)</th>
<th>P. RESIDUE (B. QUADRANT)</th>
<th>D. RESIDUE (T. QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YONE</td>
<td>0.000</td>
<td>3.007</td>
</tr>
<tr>
<td>YFOR</td>
<td>0.000</td>
<td>0.165</td>
</tr>
<tr>
<td>YTWO</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>YTHREE</td>
<td>20.001</td>
<td>0.000</td>
</tr>
<tr>
<td>YSIX</td>
<td>10.002</td>
<td>0.000</td>
</tr>
<tr>
<td>YFIV</td>
<td>10.001</td>
<td>0.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESSES (COLUMNS)</th>
<th>P. VALUE (T. QUADRANT)</th>
<th>D. VALUE (B. QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTWO</td>
<td>30.003</td>
<td>0.000</td>
</tr>
<tr>
<td>XONE</td>
<td>30.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
**PROBLEM NO. 7**

**SOLUTION IS OPTIMAL**

**NO. OF ITERATIONS = 9**

**VALUE OF OBJECTIVE FUNCTION = 699.4**

<table>
<thead>
<tr>
<th>RESOURCES (ROWS)</th>
<th>P. RESIDUE (BOT. QUADRANT)</th>
<th>D. RESIDUE (TOP QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULT1</td>
<td>0.000</td>
<td>9.665</td>
</tr>
<tr>
<td>ULT2</td>
<td>0.000</td>
<td>48.000</td>
</tr>
<tr>
<td>UHOT</td>
<td>0.000</td>
<td>48.000</td>
</tr>
<tr>
<td>UHST</td>
<td>5.829</td>
<td>0.000</td>
</tr>
<tr>
<td>HLOG</td>
<td>150.000</td>
<td>0.000</td>
</tr>
<tr>
<td>HLTH</td>
<td>43.269</td>
<td>0.000</td>
</tr>
<tr>
<td>H16X</td>
<td>26.148</td>
<td>0.000</td>
</tr>
<tr>
<td>HDR</td>
<td>553.834</td>
<td>0.000</td>
</tr>
<tr>
<td>H18X</td>
<td>17.114</td>
<td>0.000</td>
</tr>
<tr>
<td>EH16</td>
<td>30.000</td>
<td>0.000</td>
</tr>
<tr>
<td>EH18</td>
<td>20.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESSES (COLUMNS)</th>
<th>P. VALUE (TOP QUADRANT)</th>
<th>D. VALUE (BOT. QUADRANT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLTS</td>
<td>110.330</td>
<td>0.000</td>
</tr>
<tr>
<td>HP16</td>
<td>271.099</td>
<td>0.000</td>
</tr>
<tr>
<td>DR16</td>
<td>120.000</td>
<td>0.000</td>
</tr>
<tr>
<td>HDR18</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>HB16L</td>
<td>271.090</td>
<td>0.000</td>
</tr>
<tr>
<td>DR16</td>
<td>1737.600</td>
<td>0.000</td>
</tr>
<tr>
<td>SV16</td>
<td>1737.600</td>
<td>0.000</td>
</tr>
<tr>
<td>SV18</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>HL18</td>
<td>0.000</td>
<td>21.619</td>
</tr>
<tr>
<td>HP18</td>
<td>0.000</td>
<td>26.071</td>
</tr>
<tr>
<td>DROT</td>
<td>0.000</td>
<td>19.164</td>
</tr>
</tbody>
</table>
APPENDIX II

COMPUTER PROGRAMS
PROGRAM FOR WANG 200 WITHOUT DISK DRIVES

A810 REM INTRODUCTION
  :PRINT HEX(03)
  :PRINT TAB(10);"RPM LP VERSION WANG 2200"
  :PRINT
  :PRINT
A820 REM "THIS AMAZING LITTLE MINT WILL HANDLE LP PROBLEMS UPTO 200 VARIABLES AND 100 LESS THAN CONSTRAINTS.
A830 PRINT "THE PRINTER IS SLOW, AND Owing to a lack of core space the cassette has to move a lot to store and to retrieve data."
A840 PRINT "however if you follow the instructions and be patient you will get the results."
A850 LOAD "PA3"
REM -RNS-INPUTS AND CORRECTS DATA
DIM BL,A1,N
DIM X1(101),N1(101)
0020 PRINT HEX(00)
   PRINT "(0) YOU WISH TO-----"
   PRINT "(1) CREATE A NEW DATA FILE"
   PRINT "(2) MODIFY AN OLD FILE"
   PRINT "(3) RUN AN EXISTING FILE"
   PRINT "PL CHOOSE ONE OPTION"
0040 INPUT M
   IF M<3 THEN 50
   LOAD "TRANSFER"
0050 REWIND
   STOP "MOUNT DATA CASSETTE AND CONTINUE"
   IF M=1 THEN 70
   LOAD "COUNT"
   LOAD A1
   REWIND
0060 DATA LOAD "ROWS"
   DATA LOAD X1,N1
   G0|0
0070 PRINT HEX(00)
0080 DATA SAVE OPEN "ROWS"
   PRINT "INPUT CONSTRAINT NO. RESOURCE NAME AND THE RNS VALUE IN THAT ORDER. THE NAME MUST NOT EXCEED 4 CHARACTERS."
   PRINT "INPUT 1 DATA SET PER LINE. THE LARGEST NO. OF CONSTRAINT NO. ENTERED WILL DETERMINE THE TOTAL NO. OF RESOURCE NODES IN THE PROBLEM."
   PRINT "TO TERMINATE DATA ENTRY TYPE 0.0"
   INIT (0001,(0,0)N)
   BL=0
   0110 INPUT 1,N1(1+3),X
   IF T=0 THEN 130
   0120 IF X=1+3 THEN 110
   BI=I+3
   B0|0
   0130 PRINT "LISTING OF YOUR DATA IS ON OUR SUPER HIGH SPEED PRINTER"
   SELECT PRINT 212
   PRINT "NAME:1-2, NAME:1-2, NAME:1-2"
'NEXT I
0140 SELECT PRINT 805
  PRINT HEX(03)
  INPUT "DO YOU WANT TO MAKE ANY CHANGES"
  IF A<"YES" THEN 160
  PRINT "TO CORRECT YOUR ERROR REENTER THE DATA FOR ".
  PRINT "THE INCORRECT ONES USING THE SAME FORMAT AS BEFORE. END WITH 0.0.0"
  GOTO 110
0160 DATA SAVE X$O
  DATA SAVE N$O
  DATA SAVE END
  DATA SAVE OPEN "COLUMN"
  REWIND
  STOP "MOUNT PROGRAM CASSETTE AND CONTINUE"
0170 REWIND
  LOAD "VAR"

VHR
PAGE= 1
6/22/76
0140 REWIND
  STOP "MOUNT DATA CASSETTE AND CONTINUE"
  DIM X$(1914)
0160 PRINT HEX(03)
  DATA LOAD "COLUMN"
0160 IF M<2 THEN 90
0180 INIT (000)X$O
  PRINT "INPUT VARIABLE NO. VARIABLE NAME (NOT MORE THAN 4 CHAR) AND VALUE IF NO MORE VARS THEN TYPE 0.0.0"
0190 INPUT L.X$(01),X
  IF I=0 THEN 90
  M=M+1
  PRINT (14("###"))X$O FROM X
0210 PRINT "INPUT CONSTRAINT NOS AND COEFFICIENTS AT THE END OF ALL ENTRIES FOR A VHR TYPE 0.0"
0070 INPUT L.X
0080 IF I=0 THEN 80
0090 PACK (+###-###-##-##-##-##-###) FROM X
0100 GOTO 70
0110 DATA SAVE X.FO
0120 GOTO 40
0130 BACKSPACE BEG
0140 PRINT "A LISTING OF YOUR DATA IS ON THE"
0150 PRINT "PRINTER"
0160 SELECT PRINT 213
0170 FOR J=1 TO A1
0180 PRINT L.<-25,>
0190 DATA LOAD X.FO
0200 PRINT X.$(I)
0210 PRINT (4.###-###-###-###)X$(3) TO X
0220 PRINT TAB(10):"VALUE":X
0230 PRINT "COEFFICIENTS"
0240 FOR J=1 TO A1
0250 PRINT (4.###-###-###-###)X$.J TO X
0260 IF X=0 THEN 120
0270 PRINT "Y":J-3,X
0280 NEXT J
0290 NEXT I
0300 SELECT PRINT 005
0310 PRINT HEX(03)
0320 PRINT "DO YOU WISH TO MAKE ANY CHANGES"
0330 IF 0=1 THEN 240
0340 IF 0=$="NO" THEN 240
0350 PRINT "ENTER THE VARIABLE NO TO BE CORRECTED"
0360 BACKSPACE BEG

VAR

6/22/76

0400 IF 0=1 THEN 150
0410 SKIP 0=1
0420 PRINT "REENTER ALL THE VALUES OF THAT VARIABLE USING THE ORIGINAL FORMAT"
0430 INPUT L.X$(I),X
0440 PACK (+###-###-##-##-##-##-###) FROM X
0450 INPUT L.X
0460 IF I=0 THEN 190
0470 PACK (+###-###-##-##-##-##-###) FROM X
0480 GOTO 150
0490 IF REC=1 THEN 230

PAGE= 2
DATA RE SAVE X#0
SELECT PRINT 213
PRINT "CORRECTION"
GOTO 210
0200 DATA SAVE X#0
AI=1
SELECT PRINT 213
PLOT (-.25,)
PRINT "ADDITION"
0310 PRINT X#(1):
UNPACK #+########X$#(2) TO X
PRINT TAB(10): "VALUE": X
PRINT "COEFFICIENTS"
FOR J=4 TO B1
0320 UNPACK #+########X$#(J) TO X
IF Y=0 THEN 230
PRINT "Y": J-3, X
0330 NEXT J
GOTO 120
0340 BACKSPACE BEG
SETI A1
DATA SAVE END
DATA SAVE OPEN "COUNT"
DATA SAVE B1.A1
DATA SAVE END
0350 REWIND
STOP "MOUNT PROGRAM CASSETTE AND CONTINUE"
LOAD "TRANSFER"

TRANSFER
PAGE=1
6/22/76

0410 REM TRANSFER
.COM A1.B1
2DIM X$(100), Y$(100), N$(100)
REW
0420 STOP "MOUNT DATA CASSETTE AND CONTINUE"

K=1
G1=0
G4=4
DATA LOAD "COUNT"
DATA LOAD B1.A1
B1=B1+1
REW
0430 DATA LOAD "COLUMN"
IF G1=0 THEN 40
SET G1
0040 DATA LOAD X#(0)
  IF END THEN 80
  BIN(STR(Y#(K), 6, 1))=G1+1
  STR(Y#(K), 1, 4)=X#(1)
  K=K+1
  IF K=100 THEN 120
0050 FOR J=3 TO 61
  UNPACK (+###...X#(J)) TO X
  IF X=0 THEN 70
  BIN(STR(Y#(K), 6, 1))=G1+1
  BIN(STR(Y#(K), 5, 1))=J
0060 GOSUB 270
  K=K+1
  IF K=100 THEN 120
0070 NEXT J
  G1=G1+1
  G0100 40
0080 REWIND
  DATA LOAD "PONS"
  DATA LOAD X#(0)
  DATA LOAD X#(0)
  BIN(STR(Y#(K), 6, 1))=G1+1
  BIN(STR(Y#(K), 5, 1))=J
  IF K=100 THEN 120
0090 FOR J=4 TO 61
  UNPACK (+###...X#(J)) TO X
  IF X=0 THEN 110
  X=1
  BIN(STR(Y#(K), 6, 1))=G1+1
  BIN(STR(Y#(K), 5, 1))=J
0100 GOSUB 270
  K=K+1
  IF K=100 THEN 120
0110 NEXT J
  G1=G1+1
  G1100 5
  REWIND
  STOP "MOUNT PROGRAM CASSETTE AND CONTINUE"
```
K=1
IF G3=0 THEN 130
DATA LOAD "LPDATA"
GO TO 140
130 SKIP 36
DATA SAVE OPEN "LPDATA"
140 NR1 NR 10 DIM X$(100)4, N$(100)4
1=VAL(STR$(Y(K),6,1))
150 INIT (600)X$(0)
G3=G3+1
X$(1)=STR$(Y(K),1,4)
IF X$(1)<"X#" THEN 160
L=-2000
L=3
GOTO 260
160 L=11+1
K=K+1
170 IF G3<1 THEN 210
L=VAL(STR$(Y(K),6,1))
UNPACK (### ###)STR$(Y(K),4) TO X
180 IF JO3 THEN 190
X=-X
190 L=3
GOTO SUB 260
```

TRANSFER PAGE= 3

```
0350 K=K+1
0360 IF K>62 THEN 210
0370 GO TO 170
0410 DATA SAVE X$(0)
0420 IF G3<61 THEN 150
0420 IF G3<61 THEN 240
PRINT
PRINT "MOUNT DATA CASSETTE AND CONTINUE"
0470 NR1 NR 10 DIM X$(100)4, N$(100)4
0480 INIT (600)X$(0), X$(0)
K=1
0490 IF G3=81 THEN 80
0490 IF G3=81 THEN 30
0540 INIT (600)X$(0)
X=1
L=K
GOTO SUB 260
DATA SAVE X$(0)
DATA SAVE END
NR=NR+2
DATA SAVE OPEN "RNAME"
NR(O1)="2444"
```
1.0F10 PFH PIVOT- FINDS PIVOT ROW, RI=ROW, CI=COL
2.0F15 DATA 034
3.0F16 Init (03,010,030)
4.0F20 DATA LOAD DC OPEN R"DATFIL"
5.0F25 FOR I=1 TO RI-1
6.0F30 DATA LOAD DC X(0)
7.0F35 UNPACK (++ #~~~~~~~~~X(0)) TO X
8.0F40 IF X=0 THEN 40
9.0F45 IF X=X THEN 40
10.0F50 E=X
11.0F55 CI=I
12.0F60 NEXT I
13.0F65 DISCARD END
14.0F70 DISCARDSPACE 1
15.0F75 DATA LOAD DC X(0)
16.0F80 FOR J=4 TO RI
17.0F85 UNPACK (++ #~~~~~~~~~X(0)) TO X
18.0F90 IF X=0 THEN 70
19.0F95 NEXT J
20.0F9A IF I=0 THEN 160
21.0F9B GOTO 90
22.0FAB PRINT HEX(0)
23.0FAC PRINT "SOLUTION PROCESSING INFEASIBLE A
24.0FD INPUT "DO YOU WANT A LISTING OF THE TAB
25.0FD IF OR="NO" THEN 170
26.0FD1 REWIND
27.0FD2 LOAD "TAB"
28.0FD3 CI=1+1
29.0FD4 DISCARDSPACE BEG
30.0FD5 IF CI=1 THEN 160
31.0FD6 DISCARD CI-1
32.0FD7 DATA LOAD DC P(0)
33.0FD8 LB=100000
34.0FD9 FOR J=4 TO RI
35.0FD9 UNPACK (++ #~~~~~~~~~P(0)) TO E
36.0FD9 IF E<0 THEN 120
37.0FD9 UNPACK (++ #~~~~~~~~~P(0)) TO R
38.0FD9 X=R/E
39.0FD9 IF LOG X THEN 120
40.0FD9 LB=X
41.0FD9 RI=J
REM NAME NAMES BASIC VARIABLES
REM INPUT "DO YOU WANT TO SKIP PRINTING TABLE AT EACH ITERATION", MX
DATA LOAD DC OPEN R "DATFILE"
FOR I=1 TO M1-1
  DATA LOAD DC XI
  STR(XI, [2], 1)="X"
  BIN(STR(XI, [2], 4, 1))=1
BACKSPACE 1
DATA SAVE DC XI
NEXT I
FOR J=1 TO B1
  STR(P$[J], 1, 1)="P"
  BIN(STR(P$[J], 2, 4, 1))=J-3
NEXT J
RENAME "Pivot"
REM VECTOR CALCULATES THE PIVOT VECTOR

0010 DIM Z(1, 1)
0020 X=STR(P(2), 3, 1)
0030 Z=VAL(STR(P(2), 4, 1))
0040 Y=VAL(STR(R1), 2, 1)
0050 STR(P(2), 3, 1)=Z
0060 STR(R1), 1, 1)=Y
0070 B=VAL(STR(P(2), 4, 1))=Y
0080 B=VAL(STR(R1), 2, 1)=Z
0090 FOR J=1 TO B1
0100 IF J=R1 THEN 60
0110 UNPACK (+# ####-----)P(J) TO X
0120 X=#-~P1
0130 GOTO 70
0140 X=V/P1
0150 CONVERT X TO B3, (+# ####-----)
0160 IF STR(A8, 3, 1)<"5" THEN 100
0170 IF STR(A8, 3, 1)>"5" THEN 90
0180 G=RND(0) * 2
0190 IF G>=1 THEN 100
0200 Z=X
0210 Z=RND(X)+ 00005
0220 IF Z>0 THEN 100
0230 Z=X
0240 PACK (+# ####-----)P(J) FROM X
0250 NEXT J
0260 DATA LOAD DC OPEN R"DATFILE"
0270 IF C1=1 THEN 120
0280 DISK: C1=1
0290 DATA SAVE DC P1C
0300 RETURN
0310 LOAD "ITERATH"
0010 REM ITERAFN
0015 DIM X((30, 4, 0, 0)
0020 DATA N((B1), X=(B1)
0025 DATA LOAD "UPDATA"
0030 FOR K=1 TO B1
0035 DATA LOAD X(K)
0040 IF K=B1 THEN 20
0045 FOR I=1 TO B1
0050 T=0
0055 IF I=1 THEN 40
0060 UNPACK (+, #, #, #, #, #) TO X
0065 IF J=I THEN 40
0070 IF J>1 THEN 50
0075 T=T+X
0080 GOTO 50
0085 UNPACK (+, #, #, #, #, #) TO Y
0090 T=T+(Y
0095 NEXT J
0100 GOSUB 220
0105 X1(1)=X1(1)
0110 X1(1)=X1(2)
0115 NEXT 1
0120 SPACE 1
0125 DATA SAVE X(10)
0130 NEXT F
0135 PRINT C=-1000, >
0140 SELECT PRINT 213
0145 PRINT "ITERATION": II "PROCESSING"
0150 UNPACK (+, #, #, #, #, #, #, #, #) TO 0
0155 X=10 30000
0160 PRINT "PRESENT OBJECTIVE VALUE": X
0170 IF N="YES" THEN 200
0175 :SELECT PRINT 213(90)
0180 :BACKSPACE BEG
0185 IF N="NO" THEN 200
0190 :INPUT "DO YOU WANT A LISTING OF THE TAB
0195 "LENT", N
0200 IF N="NO" THEN 200
0210 N1=9
0215 N2=14*(B1-1)
0220 N4=9
0225 :PRINT <, N1=50, >
0230 N1=N1+1
0100 PRINT C.N5,>
:PRINT (N4+4); STR(X*(2), 2, 1); X
:FOR J=3 TO 81
:UNPACK (++'#####———XX&X) TO D
0150 N=0
:PRINT TAB(N4):
:PRINT USING 210, X
:NEXT J
0160 N4=N4+13
:IF N3=5 THEN 180
:GOTO 130
0170 N4=1
0180 N4=N4+13
:PRINT C.N5,>
:FOR J=4 TO 81
:N=VAL (STR(R#J, 2, 1))
:PRINT TAB(N4); STR(R#J, 1, 1); X
:NEXT J
0190 IF R3=1 THEN 290
:GOTO 120
0200 REN Down
:LOAD "Pivot"
0210 Z=1'#####———
0220 CONVERT T TO A$(++'#####———)
:IF STR(A$, 8, 1)<"5" THEN 250
0230 T=RND(T)+2
:IF T>1 THEN 250
0240 Z=T
:T=RHS(T)+ 00005
:IF Z>0 THEN 250
:T=-1
0250 PACK (++'#####———XX&X) TO FORT
:RETURN
76

0010 REM TAB
0020 REI DIM RX(1),RX(2)
0030 DATA LOAD "RNMAE"
0040 DATA LOAD RX(0)
0050 BSPACE 3F
0060 DATA LOAD "LPDAMH"
0070 SELECT PRINT 212
0080 IF 01<>0 THEN 50
0090 PLOT <,,-200,>
1000 PRINT "SOLUTION IS OPTIMAL"
0100 IF 01<>0 THEN 50
0110 PRINT "VALUE OF OBJECTIVE FUNCTION ="; RX
0120 PLOT <,,-79,>
0130 IF RX(1)>0 THEN 70
0140 PLOT <-,,-70,>
0150 IF RX(1)<0 THEN 70
0160 PLOT <-,,-70,>
0170 PLOT <,,-25,>
0180 FOR J=1 TO B1-1
0190 DATA LOAD RX(0)
0200 IF STR$(RX(2),1)<>Y" THEN 110
0210 UNPACK (++#+++++++)(X$3) TO V
0220 RX=VAL(STP(RX(2),4,1))
0230 PRINT "USING 250; R-X(X1-2;),V"
0240 NEXT J
0250 IF RX(1)<>0 THEN 110
0260 PRINT "USING 270; P$3(X+3),V"
0270 NEXT 1
0280 DATA LOAD RX(0)
0290 FOR J=1 TO B1
0300 IF STR$(RX(J),1)<>Y" THEN 140
0310 UNPACK (++#+++++++)(X$J) TO V
0320 RX=VAL(STP(RX(J),2,1))
0330 IF RX(X3)="Y" THEN 140
0340 PRINT "USING 280; P3(X+3),V"
0350 NEXT J
0360 PLOT <,,-50,>
0100 DATA LOAD R#O
  IF P#(1)="####" THEN 200
  PRINTUSING 270 , P#(1), V
  GO TO 200
0190 IF X#1 THEN 200
  PRINTUSING 270 , X#(1), V
0000 NEXT 1
  STOP END
  BACKSPACE 1
  DATA LOAD R#O
  FOR J=4 TO 81
0210 IF STR$(P#(1),1,1)="X" THEN 240
  UNPACK (###-####-####-####-####) TO V
0260 X=VAL(STR$(P#(1),2,1))
  BACKSPACE BEG
  IF X#1 THEN 230
  STOP X=1
0230 DATA LOAD P#O
  IF P#(1)="####" THEN 240
  PRINTUSING 280 , P#(1), V
0340 NEXT J
  IF 0=1 THEN 220
  IF 0=2 THEN 230
  IF 0=3 THEN 260
  IF 0=4 THEN 310
0350 X#(SOURCES,COLUMNS) P. RESIDUE
  D. RESIDUE
  X#(SOURCES,COLUMNS) BUT quadrant
  (TOP QUADRANT)
0360 X#(PROCESSES,COLUMNS) P. VALUE
  D. VALUE
  X#(SOURCES,COLUMNS) (TOP QUADRANT)
0279 X     ####
            0.000
0286 X     ####
            -####.####
0000
0299 REWIND
        LOAD "PIVOT"
0300 PRINT "SOLUTION PROCESSING INFEASIBLE"
        GOTO 320
0310 PRINT "UNBOUNDED SOLUTION DETECTED"
0320 REWIND
    PRINT "IF YOU WISH TO MAKE CHANGES IN THE DATA, CLEAR THE MACHINE AND START AGAIN"
0330 REWIND
    END
PROGRAM FOR WANG 2200 EQUIPPED WITH DISK DRIVES.

TRANSFER
8110 FOR K=10 TO 4, P=10 D4, R=10 D4, M=10 D4
9211 IF K=10 THEN "DO YOU WANT TO SKIP PRINTING TAB LEADS AT EACH SECTION", M2
9220 DATA LND "LPDATA"
9230 FOR I=1 TO AL-1
9240 DATA LND X=0
9250 ST[STC0D, 2, D5]="X"
9260 ENDST[STC0D, 2, D5]=1
9270 BSPACE 1
9280 IF I=4 THEN X=0
9290 NEXT I
9300 FOR J=4 TO B1
9310 ST[STC0D, 1, D5]="Y"
9320 ENDST[STC0D, 2, D5]=1-J-25
9330 NEXT J
9340 END
9350 END "PIVOT"

PIVOT
1600 DATA LOAD PRO
1601 REIN
1602 FORM 3696
1603 E3:1 TO E
1604 UNPACK @34443000) TO E
1605 UNPACK @45443000) TO K
1606 X:WE
1607 IF E3 THEN 130

1120 IF E4 THEN 170
1130 IF E4="NO" THEN 170
1140 PRINT "DO YOU WANT A LISTING OF THE TABLEAU?"
VECTOR
6/22/76

0100 PRINT "VECTOR CALCULATES THE PIVOT VECTOR"
0110 DIM X(1:4)
0120 PRINT "DIM X(1:4)"
0130 V1=STR(P(2), 3, 1)
0140 S1=STR(RD, 1, 1)
0150 Y=VAL(STR(922(2), 4, 1))
0160 OR=STR(RD, 2, 1)
0170 S2=STR(922(2), 3, 1)=0
0180 STR(RD, 1, 1)=S2
0190 STR(RD, 1, 1)=S2
0200 FOR J=1 TO 81
0210 IF J=1 THEN 60
0220 X=INT(S1+"00000") TO X
0230 GOTO 70
0240 Y=1+1
0250 CONVERT X TO RL (###) "####"
0260 IF STR(RD, 1, 1)="5" THEN 160
0270 IF STR(RD, 1, 1)="5" THEN 90
0280 IF R=1 THEN 160
0290 Z=S
0300 X=ABS(X)+ 00005
0310 IF Z=0 THEN 160
0320 X=X
0330 PRINT (###) "####"
0340 X=INT(X)
0350 DATA LOAD "UPDATA"
0360 IF CI=1 THEN 120
0370 SKIP CI=1
0380 DATA RE SAVE P=0
0390 RESTART
0400 DATA LOAD "ITERP"
REM IITERATN
:DIM X$(61), D$(4)
:DATA LOAD DC OPEN R"DATAFILE"

FOR I=1 TO 61
:DATA LOAD DC X$(I)
:IF K=I THEN 70
:FOR I=3 TO 61
:TM
:FOR J=3 TO 61
:UNPACK (+# ######)X$(I) TO K
:IF J=K1 THEN 40
:IF J<K THEN 50
:T=T+X
:GTO 50
:UNPACK (+# ######)X$(I) TO Y
:T=T+(X*Y)
:NEXT J
:NEXT I
:BACKSPACE 1
:DATA SAVE DC X$(I)

NEXT K
:PRINT HE(000000000000)
:SELECT PRINT 215
:PRINT "ITERATION": I1 "PROCESSING"
:UNPACK (+# ######)X$(I) TO D
:NEXT I
:X=D-3000
:PRINT "PRESENT OBJECTIVE VALUE":X

IF M="YES" THEN 200
:SELECT PRINT 215(90)
:BACKSPACE 1
:SELFCT PRINT 2172: (90)
:PRINT 4-SPACE BEG

IF M<"NO" THEN 200
:INPUT "DO YOU WANT A LISTING OF THE TAB
:LEAP", O$;

IF O$="NO" THEN 200
:N1=0
:N3=14*(B1-1)
:N4=0
:PRINT C,-N3-N4>
:N1=N1+1
:DATA LOAD DC X$(I)
:IF END THEN 170
DATA LOAD X=0
IF N=1 THEN 170
X=X+1 STRC$(2,2,1)
140 PRINT N, N
PRINT TAB(N4+4):STR$(2,2,1):X
FOR J=3 TO 61
UNPACK (+##+##+##+##+##)X$::J TO D
150 NEXT J
PRINT TAB(N4):
PRINT USING 210,X
NEXT J
160 N4=N4+13
IF N4=5 THEN 180
GOTO 130
170 N2=N1-1
180 N1=144-(B1-3)
PRINT N, N
FOR J=4 TO 61
X=VAL(STR$(2,J,1))
PRINT TAB(N4):STR$(2,J,1):X
NEXT J
190 IF A9=1 THEN 200
GOTO 120
200 REIND
LOAD "P1901"
210 1=$##+##+##+##
220 CONVERT 1 TO 65, (+##+##+##+##)
IF STR$(A5,3,1)="5" THEN 250
230 C=RND(0)+2
IF C=1 THEN 250
240 C=1
T=A25(C)+INT(0.99995
IF T<250 THEN 250
T=C
250 PACK (+##+##+##+##)N1$(1)FROMT
RETURN
0010 REM TAB
0015 DATA LOAD DC OPEN R "RNAME"
0020 DATA LOAD DC P#0
0025 DATA LOAD DC OPEN R "DATFILE"
0030 SELECT PRINT 215(55)
0035 IF 0104 THEN 50
0040 PRINT HEX(OBABA0A80A)
0045 PRINT "SOLUTION IS OPTIMAL"
0050 OF ITERATIONS =";I1
0055 PRINT HEX(OBABA0A80A)
0060 :DSHIF END
0065 :DARKSPACE 1
0070 DATA LOAD DC X#0
0075 UNPACK (#########)X#(2) TO Y
0080 X=Y-3000
0085 :PRINT "VALUE OF OBJECTIVE FUNCTION =";X
0090 PRINT HEX(OBABA0A80A)
0095 PRINT USING 255
0100 PRINT USING 255
0095 PRINT HEX(OBABA0A80A)
0105 :DARKSPACE BEG
0110 FOR I=1 TO P1-1
0115 :DATA LOAD DC X#0
0120 IF S#R(X#(2),3,1)<>"Y" THEN 110
0125 UNPACK (#########)X#(2) TO Y
0130 X=VAL(S#R(X#(2),4,1))
0135 IF P#R(X+3)="Y#" THEN 110
0140 PRINT USING 270,P#R(X+3),Y
0145 NEXT I
0150 DATA LOAD DC X#0
0155 FOR J=4 TO Y1
0160 IF S#R(X#(4),1,1)<>"Y" THEN 140
0165 UNPACK (#########)X#(4) TO Y
0170 X=VAL(S#R(X#(4),2,1))
0175 IF P#R(X+3)="Y#" THEN 140
0180 PRINT USING 270,P#R(X+3),Y
0185 NEXT J
0190 PRINT HEX(OBABA0A80A)
0195 PRINT USING 255
0200 PRINT USING 255
0195 PRINT HEX(OBABA0A80A)
0205 :DARKSPACE BEG
0210 FOR I=1 TO P1-1
0215 :DATA LOAD DC X#0

THE

IF STATEMENT (J = 1, 3, 5, 7, 9, 11) THEN 10

IF (x) THEN 228

IF (y) THEN 240

IF (z) THEN 249

IF (t) THEN 250

IF (u) THEN 251

IF (v) THEN 252

IF (w) THEN 253

IF (x) THEN 254

IF (y) THEN 255

IF (z) THEN 256

IF (t) THEN 257

IF (u) THEN 258

IF (v) THEN 259

IF (w) THEN 260
APPENDIX III

WANG 2200 SPECIFICATIONS
SPECIFICATIONS

CRT (Cathode Ray Tube) - Model 2216

Unit Size
- Height: 14 in. (35.6 cm)
- Depth: 16 in. (40.6 cm)
- Width: 21 1/2 in. (54.6 cm)

Display Size
- Height: 8 in. (20.3 cm)
- Width: 10 1/2 in. (26.7 cm)

Capacity
- 16 lines, 64 characters/line

Character Size
- Height: 0.20 in. (0.51 cm)
- Width: 0.12 in. (0.30 cm)

Weight
- 36 lbs (16.3 kg)

System 2200 Power Requirements
- 115 VAC or 230 VAC ± 10%
- 50 or 60 Hz ± 1/2 cycle

System 2200 Operating Environment
- 50°F to 90°F (10°C to 32°C)
- 40% to 60% relative humidity

Tape Drive - Model 2217

Stop/Start Time
- 0.09/0.05 sec

Capacity
- 522 bytes/ft (1712 bytes/m)

Recording Speed
- 7.5 IPS (19.05 cm/sec)

Search Speed
- 7.5 IPS (19.05 cm/sec)

Transfer Rate
- 326 characters/sec (approx.)

Inter-record Gap
- 0.6 in. (1.52 cm)

(Capacity and transfer rate include gaps and redundant recording.)

CPU (Central Processing Unit) - System 2200, Model A or B

Built-in Functions
- Mathematical & Trigonometric Functions
  - EXP: e to the power of x
  - LOG: Natural Log
  - SQR: Square Root
  - π: Pi
  - SIN: Sine
  - COS: Cosine
  - TAN: Tangent
  - ARCSIN: Inverse Sine
  - ARCCOS: Inverse Cosine
  - ARCTAN: Inverse Tangent
  - RND: Random Number Generator
- Logical & Data Manipulation Functions
  - ABS: Absolute Value of a Number
  - INT: Integer Value of a Number
  - 1, 0, or +1 if a number is negative, 0, or positive.
  - STR: Selection of one or more characters in an alphanumeric string.
  - HEX: Hexadecimal Values
  - LEN: Length of Alphanumeric Variable

Variable Formats
- Scalar Numeric Variable,
- Numeric 1- and 2-dimension Array Variables,
- Alphanumeric String Variable,
- Alphanumeric 1- and 2-dimensional String Arrays.

Average Execution Times (Milliseconds)

Add/Subtract: 0.8
Multiply/Divide: 3.87/7.4
Square Root/eX: 46, 4/25.3
Loge x, XY: 23, 2/45.4
Integer/Absolute Value: 0, 24/0.02
Sign/Sine: 0, 25/38.3
Cosine/Tangent: 38, 9/78.5
Arctangent: 72.5
Read/Write Cycle: 1.611 sec

(Average execution times were determined using random number arguments with 13 digits of precision. Average execution times will be faster in most calculations with arguments having fewer significant digits.)
### SPECIFICATIONS (Cont.)

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Size</td>
<td>34 lbs (15.4 kg)</td>
</tr>
<tr>
<td>Peripheral Capacity</td>
<td></td>
</tr>
<tr>
<td>Dynamic Range</td>
<td></td>
</tr>
<tr>
<td>Subroutine Stacking</td>
<td>6 (expandable to 11 max)</td>
</tr>
</tbody>
</table>

| Peripheral Capacity | 4,096 bytes (expandable to 32K) |
| Dynamic Range | 10^-99 to 10^+99 |
| Subroutine Stacking | No Limit |

| KEYBOARD | 
|----------|--------|
| Model 2215 |  
| Height | 3 in. (7.62 cm) |
| Depth | 10 in. (25.4 cm) |
| Width | 17 1/2 in. (44.5 cm) |
| Weight | 7 lbs (3.2 kg) |

| Model 2222 |  
| Height | 3 in. (7.62 cm) |
| Depth | 10 in. (25.4 cm) |
| Width | 19 1/2 in. (49.5 cm) |
| Weight | 7 1/2 lbs (3.4 kg) |

*Trigonometric arguments in radians, degrees or gradians

Wang Laboratories reserves the right to change specifications without prior notice.