CLEARINGHOUSE FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION, CFSTI
DOCUMENT MANAGEMENT BRANCH 410.11

LIMITATIONS IN REPRODUCTION QUALITY

Accession #

☑ 1. We regret that legibility of this document is in part unsatisfactory. Reproduction has been made from best available copy.

☐ 2. A portion of the original document contains fine detail which may make reading of photocopy difficult.

☐ 3. The original document contains color, but distribution copies are available in black-and-white reproduction only.

☐ 4. The original distribution copies contain color which will be shown in black-and-white when it is necessary to reprint.

☐ 5. The processing copy is available on loan at CFSTI.

☐ 6.

TSL-107-8/64 Processor....

MANUAL FOR THE RAND-IBM CODE FOR LINEAR PROGRAMMING ON THE 704

William Orchard-Hays
Leola Cutler
and
Harold Judd (I.B.M.)

P-842

May 16, 1956

The RAND Corporation
1700 MAIN ST. SANTA MONICA CALIFORNIA

51 p.

Best Available Copy

LINEAR PROGRAMMING

by Wm. Orchard-Hays (Rand Corp.)
Hal Judd (I.B.M.)
Leola Cutler (Rand Corp.)

INTRODUCTION

A linear programming system which is discussed is the modified simplex procedure with the product form of inverse. It is designed to solve the classical linear inequalities problem and most of its variations on the IBM 704.

MACHINE REQUIREMENTS

The basic machine needed for this system is the IBM 704 with 4 logical drums, 4 tapes and 4096 words of magnetic core storage. One additional tape may be used optionally for storing output results. One tape may be used for input to the data assembly which is run prior to using the system. Hence the input is from cards or tape and the output is the on-line printer or off-line printer. The card punch is used by the data assembly and also for punching restart information.

NOTATION

The quantities displayed in a matrix require two indices, one for row and one for column. Here a row vector will be denoted by a general subscript, one of its elements by a specific subscript, as:

c_j is a row vector

c_3 is the element of index 3.

A column vector will be denoted by a general superscript, one of its elements by a specific superscript, as:

b^j is a column vector

b^5 is the element of index 5.

A matrix will be denoted by both a subscript and a superscript, as:

a_j^i is a matrix

a_j^5 is the row consisting of the element of index 5 in each column.

a_4^i is the column consisting of the element of index 4 in each row.

a_2^6 is the element in row 6 and column 2.

The ranges of values will be specified when the index is first mentioned.

Note that a^i is the transpose of a_i. The transpose of a ma ix must be denoted by defining a new letter, e.g., $b_j^i = a_i^j$ defines b_j^i as the transpose of a_j^i.

For a given matrix of coefficients a_j^i and a column of constants b^i, then, a system of m simultaneous linear equations in n unknowns x^j will be written

$$a_j^i \, x^j = b^i \qquad (i = 1, \ldots, m; \;\; j = 1, \ldots, n)$$

instead of $\sum_{j=1}^{n} a_j^i \, x^j = b^i$ for $i = 1, \ldots, m$. This summation convention is used whenever a column is multiplied on the left by a row. A matrix may be multiplied on the left by a row vector to produce a row vector:

$$c_i \, a_j^i = d_j \, .$$

A matrix may be multiplied on the left by a row vector and on the right by a column vector to produce a scalar:

$$c_i \, a_j^i \, x^j = z \, .$$

Finally, of course, a matrix may be multiplied on either side to produce another matrix by multiplying by a matrix with the proper dimensions:

$$a_j^i \, b_k^j = c_k^i$$

$$d_i^h \, a_j^i = e_j^h \, .$$

Uppercase letters (except T) will be used for sets. The use of letters for indices is as follows:

h, i, j, k	for general indices
ℓ, m, n	for limits
p, q, r, s	for specific indices.

When it is desired to index a quantity over time, i.e. iterations, the index will be enclosed in parentheses, e.g., $\alpha_{s(t)}^i$ is produced from $\alpha_{s(1)}^i$ after t-1 transformations. The capital letter T will be used as the current limit for t, that is, T is the current iteration number and t = 1, 2, ..., T.

STATEMENT OF PROBLEM

We define the standard linear programming problem in two forms. This is because it is mandatory to start with the identity matrix as the initial basis in all circumstances. Certain deviations from these standard forms are discussed in the detailed write-up.

STANDARD FORM 1

Given: A row of coefficients of a linear form to be optimized

$$a_j^o \quad (j = 1, \ldots, \ell)$$

A matrix of restraint coefficients

$$a_j^i \quad (i = 1, \ldots, m; \; j = 1, \ldots, \ell)$$

A column of constants

$$b^i \quad (i = 1, \ldots, m)$$

To find: A column of values for $x^j \geq 0$ $(j = 1, \ldots, \ell)$ such that the variable x^o is maximized subject to

$$(1.1) \qquad x^o + a_j^o x^j = 0 \qquad (j = 1, \ldots, \ell)$$

$$(1.2) \qquad a_j^i x^j \leq b^i \qquad (i = 1, \ldots, m)$$

Note that (1.1) is perfectly general since the sum $a_j^o x^j$ may be minimized or maximized merely by changing the signs of the a_j^o. To convert (1.2) to equalities, we define the variables $x^{\ell+i} \geq 0$ $(i = 1, \ldots, m)$. Then (1.2) becomes

$$(1.2') \qquad a_j^i x^j + x^{\ell+i} = b^i .$$

If δ_h^i $(i = 0, 1, \ldots, m; \; h = 0, 1, \ldots, m)$ is the identity matrix of order m+1 (essentially the Kronecker delta), then we can define

$$a_o^i = \delta_o^i \quad \text{and} \quad b^o = 0$$

$$a_{\ell+h}^i = \delta_h^i \quad \text{for } h = 1, \ldots, m$$

(thus defining $j = 0$ and $j = \ell+1, \ldots, \ell+m = n$) and replace both (1.1) and (1.2') by

$$(1.3) \qquad a_j^i x^j = b^i \quad (i = 0, 1, \ldots, m; \; j = 0, 1, \ldots, n).$$

If the restraint equations can be put in the form (1.2'), then some of the b^i

are permitted to be negative, if necessary. However the number of negative b^i should be small to prevent an excessive number of iterations.

If the restraints cannot be specified exactly in the form (1.2) without an excessive number of negative b^i, then the problem should be put in standard form 2. It should be noted that the slack vectors a^i_{l+h} introduced above are legitimate. The limit n will be used consistently for the number of legitimate vectors in the system (besides a^i_o which is an operational device.)

STANDARD FORM 2

Given: A row of coefficients of a linear form to be optimized

$$a^o_j \quad (j = 1, \ldots, n)$$

A matrix of restraint coefficients

$$a^i_j \quad (i = 2, 3, \ldots, m; \ j = 1, 2, \ldots, n)$$

A column of constants

$$b^i \geq 0 \quad (i = 2, 3, \ldots, m)$$

To find: A column of values for $x^j \geq 0$ $(j = 1, \ldots, n)$ such that the variable x^o is maximized subject to

$$x^o + a^o_j x^j = 0 \quad (j = 1, \ldots, n) \tag{2.1}$$

$$a^i_j x^j = b^i \quad (i = 2, \ldots, m) . \tag{2.2}$$

We now add to the system, artificially, the identity matrix (except δ^i_o which is always there) and auxiliary variables x^{n+k} $(k = 1, \ldots, m)$ and construct an auxiliary form in which the variable x^{n+1} is to be maximized first, i.e. before maximizing x^o. This process is called Phase I.

The purpose of Phase I is to eliminate the artificially added columns from the system or at least to make sure that the corresponding variables become zero.* Hence we put a weight of unity on each one in the row of index 1, which will become the auxiliary form. Let $a^i_{n+1} = \delta^i_1$ and for $k = 2, \ldots, m$

---

* At least one artificial column, usually the δ^i_1 column, must remain in the solution at zero level even while maximizing the variable x^o.

let $a^i_{n+k} = \delta^i_1 + \delta^i_k$. The variables x^{n+k} are to be non-negative except for x^{n+1} which is defined by

$$(2.3) \qquad x^{n+1} + \sum_{k=2}^{m} x^{n+k} = 0.$$

Clearly $x^{n+1} \leq 0$ and if $x^{n+1} = 0$, then all $x^{n+k} = 0$. When (and if) this condition is attained, the artificial variables are maintained at zero, including x^{n+1}, by considering (2.3) as a restraint equation while maximizing x^o (Phase II.) If x^{n+1} cannot be driven to zero, there is no solution to (2.2), $x^j \geq 0$. This condition is called Terminal 1.

The problem at this point can be displayed in the following augmented form where the x^j are shown above the matrix of detached coefficients.

$$(2.4) \qquad \begin{array}{cccccccccc} x^o & x^1 & \dots & x^n & x^{n+1} & x^{n+2} & x^{n+3} & \dots & x^{n+m} \end{array}$$

$$\begin{bmatrix} 1 & a^o_1 & \dots & a^o_n & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & 1 & \dots & 1 \\ 0 & a^2_1 & \dots & a^2_n & 0 & 1 & 0 & \dots & 0 \\ 0 & a^3_1 & \dots & a^3_n & 0 & 0 & 1 & \dots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & & \cdot \\ 0 & a^m_1 & \dots & a^m_n & 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b^2 \\ b^3 \\ \cdot \\ \cdot \\ \cdot \\ b^m \end{bmatrix}$$

Since the identity matrix still does not appear in (2.4), we must make a simple preliminary transformation. First note, however, that setting

$$(2.5) \qquad \begin{array}{l} x^j = 0 \quad \text{for} \quad j = 0, 1, \dots, n \\ x^{n+1} = -\sum_{i=2}^{m} b^i \quad \text{and} \quad x^{n+i} = b^i \quad \text{for} \quad i = 2, 3, \dots, m \end{array}$$

provides an initial solution in which all variables are non-negative except x^{n+1}. This solution will remain valid if we subtract all equations of index 2 through m from the auxiliary optimizing form, row 1. Then defining anew

$$(2.6) \quad \begin{array}{l} a_j^1 = -\sum_{i=2}^{m} a_j^i \quad \text{for } j = 1, 2, \ldots, n; \quad b^1 = -\sum_{i=2}^{m} b^i . \\ a_{n+k}^1 = 0 \quad \text{for } k = 2, 3, \ldots, m \end{array}$$

the row 1 equation takes the form

$$(2.7) \qquad a_j^1 x^j + x^{n+1} = b^1 \quad (j = 0, 1, \ldots, n).$$

Furthermore, the columns a_o^i, a_{n+1}^i, a_{n+2}^i, ...,., a_{n+m}^i now form δ_h^i.

The data assembly program computes a_j^1 and b^1 automatically. If some, but not all, of the columns of δ_h^i occur in a_j^i ($j \leq n$), then these columns should be indicated as being in the initial basis. The data assembly will then omit the corresponding rows in the sums (2.6). This is equivalent to adding artificial columns a_{n+k}^i only for those columns of δ_h^i which are missing.

The vectors a_o^i, a_{n+k}^i are never entered with the data. They are implicit in the code and, once eliminated, cease to exist. The basis headings, which are the j-indices of the columns a_j^i in position h = 0, 1, ..., m of the basis, are left zero for all artificial vectors since the position h identifies them sufficiently, i.e. they are never moved around in the basis. Legitimate columns of δ_h^i may go out of the basis and come back in out of position. Hence they must have names, as in Standard Form 1.

<u>Note</u>: The row indices i of a_j^i must have the numerical values 0, 1, ..., m, but the column indices j = 1, ..., n are used above only for expository purposes. These columns can be identified by any n distinct symbols, of five or less Hollerith characters each, which suit the fancy of the formulator of the problem. One convention has been adopted: legitimate columns of δ_h^i are denoted by UP001, UP002, ..., for "Unit Positive" vector. This convention is essential only to the re-inversion code.

INPUT

The data assembly for the linear programming system is designed to be executed independently of the main code. The options for loading data from tape or from the card reader, to load matrix elements which are punched in standard form or in a fixed field, or for punching out the matrix on binary cards are not controlled by sense switches. Instead, each bit of a word in storage, the SENSES word, is used to control these functions. In particular, the first three bits correspond to the three options noted above. A zero would correspond to the switch being up, and a one to the switch being down. A binary card must be inserted immediately following the transfer card in the data assembly deck with the SENSES word punched in the 9's left row. If the data is loaded from the card reader, a one is punched in position S of the SENSES word, i.e. a 9-punch in column 1 of the card.

PROBLEM IDENTIFICATION

The name of the problem or any suitable identification is punched in columns 2 through 72 in Hollerith characters for the heading card. This information is punched in a binary card along with other data for use by the main code which prints the information, as originally punched, as the first line on all of the output listings.

PROBLEM PARAMETERS

The essential parameters needed are punched in the parameter card in normalized form for ease in key punching, e.g., m equal to 77 would be punched in columns 1 and 2 while m equal to 177 would be punched in columns 1, 2 and 3. If the numbers are not in normalized form on the card, leading zeros must be punched. The parameter card is as follows:

cols	1 - 10	m	(number of restraint equations) max m = 255
''	11 - 20	z	(number of Phase I's desired)
''	21 - 30	q	(total number of Phases)
''	31 - 40	Σ	(index of sum row)
''	41 - 50	τ	(number of preliminary transformations)

If the sum row is computed prior to assembly and loaded with the other elements, Σ should be left blank. This is necessary if more than one Phase I is desired since there will be as many auxiliary forms as there are Phase I's.

If there is no sum row, leave Σ blank, not zero.

BASIS HEADINGS

The basis headings are punched 7 to a card using as many cards as needed. Ten columns are allotted for each heading (j) with its column index (i). The column index must be in normalized form or leading zeros must be punched. The cards are punched as follows:

cols.1 - 3	i	(column index for basis heading)
col. 4	blank or minus	(a minus sign will cause the main code to treat column i of the initial basis as an artificial unit vector, and then to bring the specified vector j into the basis arbitrarily. The number of negative basis headings must agree with parameter τ.)
cols.5 - 9	j	(5 Hollerith characters to denote column i of the identity matrix. These unit vectors must be denoted t. UP001, UP002, etc. for the reinversion code.)
col.10	blank	
cols. 11 - 20	as cols. 1 - 10	
. . . .		
cols. 61 - 70	as cols. 1 - 10	

If one of the basis headings is punched erroneously, it can be overwritten without repunching the entire card by succeeding it with a correctly punched version of the heading in error. A heading of zero will be assigned for all i's not specified.

RIGHT HAND SIDE

The right hand side vector b^1 is preceded by a card with FIRST B punched in Hollerith characters in columns 1 - 7, followed by element cards for non-zero entries as follows:

cols. 7 - 9 i (row index in normalized form or with leading zeros) col. 10 sign of element (blank is interpreted as positive)

col.11→ The integral part of the number is punched in col.11 on depending on the number of columns needed (8191 is the maximum integer allowed). The integer is separated from the fraction by a blank column. Then, the fractional part is punched using as many columns as needed (a maximum of seven columns may be used). The remaining columns on the card are left blank. For example, .0425 would have 0425 punched in cols. 12-15. Also, 10.0 would have 10 punched in cols. 11-12 only.

An alternative method has been provided to load data which is punched in fixed columns as described below. A one is punched in position 1 of the SENSES word if data is to be loaded in this form.

cols. 11-14 integral part of number
col. 15 blank
cols. 16-21 fractional part of number

If a supplementary right hand side c^1 is to be loaded, a card with NEXT B punched in columns 1-6 precedes the element cards which are in the same form as the elements of b^1.

A FIRST B card must always be used even if, for some reason, no b^1 are entered, e.g. when assembling a supplementary right side or a new matrix.

RESTRAINT MATRIX

A card with MATRIX punched in columns 1-6 precedes the element cards of the matrix a_j^1. If no matrix is to be loaded, this step is bypassed by loading a card with NO MATRIX punched in columns 1-9. The non-zero elements of the matrix a_j^1 are loaded one element per card as follows:

col. 1 always blank (corresponds to sign column in basis heading cards, but must be plus here.)

cols. 2-6 j (a five Hollerith character symbol to identify the column). All of the elements of each column vector must be in succession.

cols. 7→ row index and element punched in the same form as elements of b^1.*

A card with EOF punched in columns 1-3 will terminate loading of the matrix. A card with EOR in columns 1-3 will cause the data assembly to write the preceding vectors on the tape and start a new record with the succeeding vectors. If position 2 of the SENSES word is a one the data assembly punches the matrix in binary. When it is loaded by the main code it is possible to load part, but not all, of the vectors if they have been separated by the EOR breakpoint (see Deviations from the standard forms.) Also, a card with CURTAIN punched in columns 1-7 may be loaded between any two vectors (see Deviations from the standard forms.)

* If a sum row is used for Phase I, then the sum of each column must be less than 8192. This does not apply to the b^1.

DETAILED WRITE-UP

The codes whose use are described herein are designed to provide a complete system for solving linear programming and related problems where no special assumptions are made regarding the structure of the model. Since there are so many ramifications to such a general computer program, even for one type of problem, it is deemed necessary to include a considerable amount of explanatory material. In what follows, a general familiarity with the simplex method will be assumed. For example, no proofs will be given for the standard theorems concerning basic solutions or the simplex criterion. However, the algorithm using the product form of inverse and certain special features built into the codes will be developed in detail.

The subject matter will be divided into the following sections.

I - ADVANTAGES OF THE PRODUCT FORM OF INVERSE

In order to appreciate the advantages of this method, it is necessary to keep in mind the following facts concerning the matrix of coefficients of a typical linear programming problem and the requirements of the simplex method.

(A) The matrix is usually very sparse, the percentage of non-zero elements being typically 5 to 15 per cent of the total number.

(B) The given data is almost invariably expressed with a very few significant figures and is easily scaled to fit in a restricted range of fixed-point numbers.

(C) Some problems have a large number of variables although the number of restraints is not excessive. The ratio of number of variables to number of restraints is often between 5 to 1 and 10 to 1 and actual problems have been run with ratios as high as 30 to 1.

(D) If m is the number of restraint equations, then the essential manipulations of one cycle of the simplex method involve a change of basis in Euclidean m-space, where two successive bases differ only by one column vector. The rules of elimination in effecting this transformation are the same no matter what particular algorithm is used. Although such a transformation is easily represented, its total effect on the numerical representation of a set of vectors, i.e. a matrix, may be extensive. Furthermore, the validity of each cycle performed depends on the accuracy of all preceding cycles.

With these observations in mind, the advantages of the present method can be stated as follows:

(1) Since the original data matrix is not transformed from iteration to iteration, it is clear from (A) and (B) that an elaborate organization of the data can be performed at the outset, so that it is in the most convenient and compact form for use throughout the problem. Since the original matrix is referred to only once during the cycle, it can be stored on tape, out of the way, and its compact form minimizes transfer time as well as computing time.

(2) It is clear from (C) that it would be very expensive in some problems to transform the whole matrix on each cycle since, even if the ratio

of variables to constraints is, say, only as much as 5 to 1, it is still very unusual for the process to take 5m iterations. In other words, many vectors (activities) are never selected at all and hence there is no use doing anything to them.

(3) If a full inverse were maintained, it would be necessary to store it by rows or else linear combinations of rows would not be readily available. Even then, one would require storage for three vectors simultaneously in transforming a column. In changing basis, the transformation vector is a column and hence the modification and re-recording of the m^2 elements would be awkward and time-consuming, especially if zeros are deleted. On the other hand, the product form requires the recording of only one additional column (plus an index) on each iteration. These columns will almost surely contain many zeros and may be condensed since they are always applied column-wise and recursively to a single vector.

(4) The product form is extremely amenable to modifications of the method or for generating additional side information, since the inverse or its transpose are equally easy to apply.

II - REDUCING THE NUMBER OF TERMS IN THE PRODUCT

One apparent disadvantage of the product form is that, while only one new column is recorded each cycle, it is necessary to read T columns to apply this form where T is the number of iterations already performed. Thus, after m iterations, where m is the number of restraints, one must read more information than in reading a full inverse (not taking condensation into account.) This is still profitable for something over 2m iterations as can be seen as follows.

With an explicit inverse, we must read it twice and write it once each iteration, giving 3mT columns handled on T iterations.

With the product form, we must read t-1 columns twice on iteration t and write one new column, giving

$$\sum_{t=1}^{T} 2(t-1) + 1 = T(T-1) + T = T^2 \text{ columns handled on T iterations.}$$

Setting $T^2 = 3mT$ gives $T = 3m$. However something must be allowed for the fact that less arithmetic accompanies the writing than the reading. This disadvantage is more apparent than real, however. After, say, 2m iterations, round-off error will begin to be noticeable on large problems no matter which form is used. Many problems are solved in 2m iterations or less but, if it takes more, one can re-invert the basis matrix, at any time, producing not more than m columns of informations. The time for this inversion is much less and the accuracy of the resulting transformations is as good or better than after the same number of full simplex cycles. (The order of elimination for inverting is designed to maintain accuracy.) A special code is provided for this purpose. It is useful for solving any system of linear equations, especially where several right hand sides are used with a sparse matrix.

It is helpful, with this form of inversion, to consider the matrix of coefficients of the restraint equations as a collection, or set, of column vectors, ignoring the fortuitous ordering given to the variables in the formulation of the model. Whatever scrambling of columns that may occur in the process, is recorded in a list of basis headings which accompany and identify the basic variables of a particular solution. This point of view is adopted throughout the present discussion.

III - DEVIATIONS FROM THE STANDARD FORMS

Some discussion of alternate ways of starting a problem is indicated. In Standard Form 1, the inequalities were converted to equalities in the usual way by adding slack vectors. The matrix then contains the complete identity matrix (not necessarily in proper column order in a_j^i) and there is no difficulty. The variable corresponding to column i of the identity matrix is set equal to b^i and the name of the variable is recorded in the i-th word of the list of basis headings. This is done automatically by the data assembly program. The problem is then ready to start with this initial solution. We consider a "solution" to consist of the list of basis headings and the values of the corresponding basic variables.

In the usual set-up of the simplex method, it is assumed that all elements of the right hand side are non-negative so as to insure that a starting solution will be feasible, i.e. non-negative. This restriction has been removed in the 704 program by incorporating a composite algorithm which will remove infeasibilities as well as optimize. (See section X below.) Thus it is possible to multiply through, by minus one, equations which have negative slacks in order to make the slack vectors positive, even though the starting solution is thereby rendered infeasible. However, it is recommended that the number of initial infeasibilities be kept small to avoid an excessive number of iterations. There is no hard and fast rule about this since different problems will react differently to the same conditions.

If a given matrix does not contain all columns of the identity matrix legitimately, i.e. as positive slacks with zero coefficients in the optimizing form, then one can use Standard Form 2 and have the code perform a Phase I. In this case, it is necessary that all elements b^i are non-negative.

Experience has led to the incorporation of still another device for obtaining initial solutions. Sometimes the formulator of the problem knows a feasible basis other than the identity. Provision is made for introducing, arbitrarily, any number of column vectors into the basis at the outset, with the machine making the decision as to which column of the basis each should occupy. If the formulator has misjudged and the specified columns produce a singular matrix, the machine prints outs (and saves) the pertinent information and stops. If the resulting matrix is non-singular but the solution is infeasible (partly negative), then the composite algorithm automatically cuts in and works toward feasibility in succeeding iterations. This device of arbitrary transformations must not be used in Phase I.

One other provision of a less absolute nature has been made. Occasionally, a model is a re-work of an older problem so that something is known about its behavior. At other times the formulator has certain insights which he would like to exploit without committing himself to absolute statements regarding feasibility or singularity. It may be possible to assemble the matrix columns in order of decreasing likelihood of use, that is, with the most likely candidates for entry into a feasible or optimal basis read in first. These can be separated from the others by a "curtain" which is equivalent to the following instruction: "If any candidate for entry into the basis is available ahead of the curtain, use it; otherwise proceed to the others." Several such curtain marks may be used. They are activated by a switch so that their use is under control of the operator. They have often reduced the number of iterations required but when used injudiciously, i.e. on a mere hunch, may have the opposite effect.

There are, of course, other devices which require no special provisions in the code, except perhaps for loading. For example, certain activities (columns) may be withheld from the machine until optimality is obtained with the others. This is another advantage of the revised method and additional data may be added at any time simply by reloading the data tape.

IV - FORMS OF ARITHMETIC

The product form of inverse is generated, recorded and applied in doubled precision (DP), floating point arithmetic. Certain operations whose results are not retained are performed in DP fixed-point for faster operation. Printed output is in fixed-point form for easy reading, with 8 digits of whole number and 8 digits of fraction. Although such precision exceeds that of the inputs (a fact to remember when interpreting results), it has been found necessary to maintain this accuracy during the running of large problems.

V - THE BASIS

Any non-singular square matrix formed from m+1 columns of a_j^i is denoted by

(5.1) $\quad \mu_h^i \quad (i = 0, 1, \ldots, m;\ h = 0, 1, \ldots, m;\ \mu_0^i = \delta_0^i)$

where it is always assumed that $\mu_0^i = a_0^i = \delta_0^i$. The matrix μ_h^i is called the basis and [illegible] from [illegible] to [illegible]. [illegible] begins the computations with [illegible]

(5.2) [illegible]

This implies [illegible] (in Standard Form 2) contains [illegible]. [illegible] columns of [illegible] in any order in [illegible], we use second-order subscripts on [illegible] other names, [illegible].

(5.3) [illegible]

After T cycles [illegible], some of the columns of $^{(T)}$ will differ from [illegible]. The columns have been [illegible] from [illegible]

(5.4) [illegible]

the [illegible] in (5.4) are not the same [illegible] in (5.3). Hence it is necessary to keep the definition of [illegible] and [illegible] with (T). These $_{h}^{(T)}$ are the basic headings and are [illegible] to Program 1.

VI - THE INVERSE OF THE BASIS

We will not be so much concerned with the basis $\mu_h^{i(T)}$ on iteration T as with its inverse which will be denoted by $\pi_h^{i(T)}$, that is

$$(6.1) \qquad \pi_k^{i(T)} \mu_h^{k(T)} = \mu_k^{i(T)} \pi_h^{k(T)} = \delta_h^i .$$

The matrix $\pi_h^{i(T)}$ never exists explicitly but is carried as a product of elementary matrices which are stored in a condensed form consisting of at most one column plus an index and identification. These columns are themselves condensed with only non-zero elements recorded. They are called transformation vectors (sometimes elimination " equations" , see (6.5)) and denoted by $\eta_r^{i(t)}$. Since the index r is itself a function of t, it should be written r(t). However this will not usually be done since it sometimes appears as a superscript in an array already indexed (t-1) and confusion would result.

As already indicated, Greek letters will be used for a basis and its inverse, and also for all (m+1)-order vectors expressed in terms of the basis. Likewise certain ratios and functions involved in decisions concerning the basis will be denoted by Greek letters.

An explanation of the generation and use of the $\eta_r^{i(t)}$ vectors is in order. Since we started with $\mu_h^{i(0)} = \pi_h^{i(0)}$ for the first iteration, we will have $\pi_h^{i(T-1)}$ on hand to start iteration T. Suppose that column $a_{s(T)}^i$ has been chosen from a_j^i to replace column $h = r(T)$ in $\mu_h^{i(T-1)}$, producing a new basis $\mu_h^{i(T)}$. Throughout the following discussion, the specific index r is to be understood as r(T). Superscripts in parentheses will always refer to the entire array to which the element belongs and are not to be understood as modifying the open superscript.

Let $\alpha_{s(T)}^i$ satisfy the equation

$$(6.2) \qquad \mu_h^{i(T-1)} \alpha_{s(T)}^h = a_{s(T)}^i .$$

Clearly $\alpha_{s(T)}^i$ can be obtained by

$$(6.3) \qquad \pi_h^{i(T-1)} a_{s(T)}^h = \alpha_{s(T)}^i .$$

Now

$$(6.4) \qquad \mu_k^{i(T)} \delta_h^k = \mu_h^{i(T-1)} \quad \text{for} \quad h \neq r(T)$$

since only column $h = r(T)$ changes. However, from (6.2) we can solve for the exceptional column.

$$(6.5) \qquad \mu_r^{i(T-1)} = \frac{1}{\alpha_{s(T)}^r} a_{s(T)}^i + \mu_h^{i(T-1)} \eta^h \qquad (h \neq r \quad)$$

where

$$(6.6) \qquad \eta^h = \frac{-\alpha_{s(T)}^h}{\alpha_{s(T)}^r} \qquad\qquad (\alpha_{s(T)}^r \neq 0 \text{ by choice}).$$

Now defining

$$(6.7) \qquad \begin{array}{ll} \eta_r^{r(T)} = \dfrac{1}{\alpha_{s(T)}^r} , \quad \eta_r^{i(T)} = \eta^i & (i \neq r) \\ \eta_h^{i(T)} = \delta_h^i & (h \neq r) \end{array}$$

we can replace (6.4) and (6.5) by

$$(6.8) \qquad \mu_k^{i(T)} \eta_h^{k(T)} = \mu_h^{i(T-1)}$$

since $a_{s(T)}^i$ becomes column $\mu_r^{i(T)}$. Clearly only the column $\eta_r^{i(T)}$ and the index $r = r(T)$ need be recorded. Now multiplying both members of (6.8) on the left by $\pi_i^{j(T)}$ and on the right by $\pi_k^{h(T-1)}$, we obtain

$$(6.9) \qquad \eta_h^{j(T)} \pi_k^{h(T-1)} = \pi_k^{j(T)} .$$

Equation (6.9) is the heart of the product form of inverse. Applying it for $t = 1, 2, \ldots, T$ and using h_t to indicate dummy indices, we obtain

$$(6.10) \qquad \pi_h^{i(T)} = \eta_{h_{T-1}}^{i(T)} \eta_{h_{T-2}}^{h_{T-1}(T-1)} \cdots \eta_{h_{t-1}}^{h_t(t)} \cdots \eta_h^{h_1(1)} .$$

Hence an equation like (6.3) implies the recursive generation of its right member by using the form of $\pi_h^{i(T)}$ given by (6.10). This is easily done as follows, using (6.3) for an example.

$$\text{Let} \qquad a_{s(T)}^{i} = \bar{\alpha}_{s(1)}^{i}$$

$$\text{form} \qquad \eta_h^{i(1)} \bar{\alpha}_{s(1)}^{h} = \bar{\alpha}_{s(2)}^{i}$$

$$\vdots$$

$$(6.11) \qquad \eta_h^{i(t-1)} \bar{\alpha}_{s(t-1)}^{h} = \bar{\alpha}_{s(t)}^{i}$$

$$\vdots$$

$$\eta_h^{i(T-1)} \bar{\alpha}_{s(T-1)}^{h} = \bar{\alpha}_{s(T)}^{i} = \alpha_{s(T)}^{i}$$

It is easy to see that

$$(6.12) \qquad \begin{aligned} \bar{\alpha}_{s(t)}^{i} &= \bar{\alpha}_{s(t-1)}^{i} + \bar{\alpha}_{s(t-1)}^{r} \eta_r^{i(t-1)} && (i \neq r = r(t-1)) \\ \bar{\alpha}_{s(t)}^{r} &= \bar{\alpha}_{s(t-1)}^{r} \eta_r^{r(t-1)} && (r = r(t-1)) \end{aligned}$$

An equally simple rule suffices for multiplying $\pi_h^{i(T)}$ by a row vector on the left. Suppose it is necessary to compute

$$(6.13) \qquad f_i \pi_h^{i(T-1)} = \pi_h^{(T)} .$$

We use the transformation vectors in reverse order to that of (6.11) :

$$\text{Let} \qquad f_i = \bar{\pi}_h^{(1)}$$

$$(6.14) \qquad \text{form} \qquad \bar{\pi}_h^{(1)} \eta_i^{h(T-1)} = \bar{\pi}_i^{(2)}$$

$$\vdots$$

$$\vdots$$

(6.14 cont.)

$$\bar{\pi}_h^{(t)} \eta_i^{h(T-t)} = \bar{\pi}_i^{(t+1)}$$

$$\vdots$$

$$\bar{\pi}_h^{(T-1)} \eta_i^{h(1)} = \bar{\pi}_i^{(T)} = \pi_i^{(T)}$$

Again it is easy to see that

(6.15)

$$\bar{\pi}_i^{(t+1)} = \bar{\pi}_i^{(t)} \qquad (i \neq r(T-t))$$

$$\bar{\pi}_r^{(t+1)} = \bar{\pi}_i^{(t)} \eta_r^{i(T-t)} \qquad (r = r(T-t))$$

VII - THE PRICING OPERATION (Choosing index s)

The row vector $\pi_i^{(T)}$ in (6.13) is called the pricing vector. In the simplest problem (Phase II, maximize x^0, no infeasibilities), $f_i = \delta_i^0$. However, in a typical Phase I, $f_i = \delta_i^1$, and in general $f_i = 1$ for several i.

The pricing vector is applied to a_j^i,

(7.1) $\pi_i^{(T)} a_j^i = d_j^{(T)}$.

(The d_j are what are called $(\pm)(c_j - z_j)$ in the original simplex method.) To choose $a_{s(T)}^i$ to bring into the basis, take

(7.2) $d_s^{(T)} = \min d_j^{(T)} < 0$.

(If the minimum is not unique, the first such index s is retained.) If all $d_j^{(T)} \geq 0$, the phase is complete. In a Phase I, this is Terminal 1; in a Phase II it is called Terminal 2 and is the point at which one usually expects to arrive and quit, i.e. the optimal solution is attained.

VIII - CHOOSING THE BASIS COLUMN TO BE REPLACED (Index r)

Let the current basis be $\mu_h^{i(T-1)}$, the current optimizing form be row p and the current solution vector be $\beta^{i(T-1)}$, i.e.

(8.1) $\beta^{i(T-1)} = \pi_h^{i(T-1)} b^h$.

We will assume the solution is feasible, that is

(8.2) $\beta^{i(T-1)} \geq 0$ for $i \geq q$

where row q is the first actual restraint equation. Normally q = p+1.

The usual criterion for choosing r(T) is to choose $\theta_{r(T)}$ as follows. Let A be the set of indices $i \geq q$ for which $\alpha^i_{s(T)} > 0$. Then

$$(8.3) \qquad \theta_{r(T)} = \min_{i \in A} \left\{ \frac{\beta^{i(T-1)}}{\alpha^i_{s(T)}} \right\} \geq 0 \; .$$

The problem of degeneracy, i.e. multiple values of $i \in A$ for which $\beta^{i(T-1)} = 0$, is disregarded except for the following rule which has proved effective for reducing round-off error. (It incidentally breaks the tie in Hoffman's examples of "cycling.") If $\theta_{r(T)} = 0$ and r(T) is ambiguous, choose r(T) so that $\alpha^{r(T)}_{s(T)}$ is the largest possible (positive) value. In case of further ambiguity, take the smallest such index.

We will modify (8.3) slightly. Let R be the set of indices $i \geq q$ for which $\beta^{i(T-1)}$ and $\alpha^i_{s(T)}$ have the same sign with $\alpha^i_{s(T)} \neq 0$. Then let

$$(8.4) \qquad \theta_{r(T)} = \min_{i \in R} \left\{ \frac{\beta^{i(T-1)}}{\alpha^i_{s(T)}} \right\} \geq 0 \; .$$

Note that (8.4) gives the same result as (8.3) as long as (8.2) holds. Degeneracy is resolved in the same way, that is, if $\theta_{r(T)} = 0$, take $\alpha^{r(T)}_{s(T)} > 0$ and max.

If no $\theta_{r(T)}$ can be chosen, that is, all ratios are non-positive and zero ratios have negative denominators, then β^p has no finite maximum. A class of solutions exist as follows:

$$(8.5) \qquad \mu_h^{i(T-1)} \left(\beta^{h(T-1)} - \theta\, \alpha^h_{s(T)}\right) + \theta\, \alpha^i_{s(T)} = b^i$$

with the value

$$(8.6) \qquad \beta^{p(T-1)} - \theta\, \alpha^p_{s(T)} \longrightarrow +\infty \text{ as } \theta \longrightarrow +\infty \, .$$

This is Terminal 3. It cannot happen in a Phase I since clearly zero is an upper bound for the variable x^{n+1}.

IX - SUMMARY OF CYCLIC OPERATIONS (One Iteration)

(9.0) Test for end of a Phase I or for arbitrary halt (external switch.)

(9.1) Determine values of f_i (discussed further below.)

(9.2) Form $\pi_i^{(T)} = f_h \pi_i^{h(T-1)}$ by (6.14).

(9.3) Compute $d_j^{(T)} = \pi_i^{(T)} a_j^i$ and choose $d_{s(T)}^{(T)} = \min d_j^{(T)} < 0$ or terminate if all $d_j^{(T)} \geq 0$.

(9.4) Compute $\alpha_{s(T)}^i = \pi_h^{i(T-1)} a_{s(T)}^h$ by (6.11).

(9.5) Choose $\Theta_{r(T)}$ by (8.4) or terminate.

(9.6) Compute $\eta_r^{i(T)}$ by (6.6,7), transform $\beta^{i(T-1)}$ to $\beta^{i(T)}$ by one step as in (6.11) and record $s(T)$ for $j_{r(T)}^{(T)}$.

(9.7) (Optional) Print results of iteration

(9.8) Condense and store $\eta_r^{i(T)}$ and $r(T)$.

(9.9) (Optional) Check solution and print

X - THE COMPOSITE ALGORITHM (Forming f_i)

Suppose that a basic solution has been obtained which is infeasible, that row p is the current optimising form and that q is the smallest row index of the actual restraint equations.

(10.1) $\mu_h^{i(T-1)} \beta^{h(T-1)} = b^i$, $h \in F$ if $h \geq q$ and $\beta^{h(T-1)} < 0$.

Suppose a vector $a_{s(T)}^i$ has somehow been chosen to bring into the basis and that (8.4) is used to determine $r(T)$. After the change of basis, the new values of β^i are

(10.2) $\beta^{r(T)} = \Theta_{r(T)} \geq 0$ (the value of $x^{s(T)}$)

(10.3) $\beta^{i(T)} = \beta^{i(T-1)} - \Theta_{r(T)} \alpha_{s(T)}^i$ $(i \neq r(T))$

Now for $i \geq q$ and $i \notin F$, (10.3) gives $\beta^{i(T)} \geq 0$. However, for $i \in F$, there are two cases.

(10.4) If $i \in F$ and $\alpha^i_{s(T)} < 0$, then $\beta^{i(T)} \geq \beta^{i(T-1)}$

(10.5) If $i \in F$ and $\alpha^i_{s(T)} \geq 0$, then $\beta^{i(T)} \leq \beta^{i(T-1)}$

Hence by (10.2) and (10.3), no infeasibilities are created. By (10.2) and (10.4), some infeasibilities are improved or removed altogether. However, by (10.5) some may be made worse. To overcome this difficulty, consider the function $\lambda = \sum_{i \in F} \beta^i \leq 0$ which is a measure of the infeasibility of a solution. We wish to maximize it to zero and hence we may replace the maximization of β^p by the maximization of

(10.6) $\sigma = \beta^p + \lambda \leq \beta^p$

provided σ is monotonically non-decreasing. When σ reaches its maximum, if $\lambda = 0$ then β^p is maximum. If $\lambda < 0$, then β^p is too great and λ must be increased without regard to decreases in β^p.

Let $f_i = 1$ if $i \in F$ or $i = p$, $f_i = 0$ otherwise. Then, as can be seen from (9.2,3,4)

(10.7) $d^{(T)}_s = f_i \alpha^i_{s(T)} < 0$.

Consequently, after change of basis, the new value of σ is *

(10.8) $\sigma^{(T)} = \sigma^{(T-1)} - \theta_{r(T)} d^{(T)}_s \geq \sigma^{(T-1)}$.

If the set F is void, then (10.8) is the same as

(10.9) $\beta^{p(T)} = \beta^{p(T-1)} - \theta_{r(T)} d^{(T)}_s \geq \beta^{p(T-1)}$

which is the usual formula for the change in the maximand.

Now, however, if all $d^{(T)}_j \geq 0$, we cannot claim Terminal 2 without checking to see whether F is void. If F is not void and all $d^{(T)}_j \geq 0$, it may be because (10.9) dominates (10.8).@ In this case, we may scale down f_p , perhaps

---

* Note that if $\beta^{i(T-1)} < 0$, then $\beta^{i(T)} \leq 0$ for $i \in F$ and $i \neq r(T)$.

@ That is, $\pi^{p(T-1)}_i a^i_j \geq \left| f_h \pi^{h(T-1)}_i a^i_j \right|$ for $h \in F$.

to zero, until such time as $\lambda = 0$. If all $d_j^{(T)} \geq 0$ with $f_p = 0$, then no feasible solution exists since $\lambda < 0$ is at a maximum.

Even though $\lambda = 0$, it is sometimes desirable to set $f_p < 1$. Of course, tolerances must be built into the code in testing $d_j \geq 0$ since, if it is sufficiently small in magnitude, it ought to be considered zero. Varying f_p has the effect of varying this built-in tolerance. Provision has also been made for setting $f_p < 0$ which causes β^p to be minimized instead of maximized. This is often convenient when experimenting with a model. The value of f_p is entered on a binary card subject to a switch.

If $\sigma < \beta^p$ is at a maximum and $f_p \neq 0$, the machine will set $f_p = 0$ and stop so that other values may be loaded if desired. If $\lambda = 0$ and $f_p = 0$, the machine will set $f_p = 1$ and stop.

The variable β^p is checked for monotonic behavior each iteration, according as $f_p > 0$ or $f_p < 0$. This test is suspended if $\lambda < 0$ or when making arbitrary transformations when the behavior of β^p is unpredictable.

XI - PARAMETRIC PROGRAMMING (PLP)

Provision can be made for parametrizing the right hand side as a linear function of $\Theta \geq 0$, i.e.

$$(11.1) \qquad a_j^i x^j = b^i + \Theta c^i$$

where, if a Phase I was used, c^i must be formed in the same manner as b^i in (5.26).

To do this, first find an optimal solution for $\Theta = 0$, say

$$(11.2) \qquad \mu_h^{i(T-1)} \beta^{h(T-1)} = b^i = b^{i(T-1)} .$$

Let

$$(11.3) \qquad -\pi_h^{i(T-1)} c^h = \gamma^i .$$

Now using γ^i in place of $\alpha_{s(T)}^i$ in (8.4), a value $\Theta_{r(T)} = \Delta\Theta$ can be found such that

(11.4) $\mu_h^{i(T-1)} (\beta^{h(T-1)} - \theta_{r(T)} \gamma^h) = b^i + \theta_{r(T)} c^i$

with

(11.5) $\beta^{i(T-1)} - \theta_{r(T)} \gamma^i \geq 0 \quad \text{for } i \geq q$

and

$$\beta^{r(T-1)} - \theta_{r(T)} \gamma^r = 0 \quad (r = r(T)).$$

The parameter θ cannot be increased by more than $\theta_{r(T)}$ with the basis $\mu_h^{i(T-1)}$ without violating (11.5) but (11.4) is an optimal feasible solution to (11.1) for this value of θ. Let

(11.6)
$$b^{i(T)} = b^{i(T-1)} + \theta_{r(T)} c^i$$
$$\beta^{i(T)} = \beta^{i(T-1)} - \theta_{r(T)} \gamma^i .$$

To increase θ further, $\mu_{r(T)}^{i(T-1)}$ must be removed from the basis and replaced with some $a_{s(T)}^i$ to form a new basis $\mu_h^{i(T)}$ so that

(11.7) $\mu_h^{i(T)} \beta^{h(T)} = b^{i(T)}$

is also an optimal feasible solution to (11.1) for the same θ. The whole process can then be repeated.

The index $s(T)$ is determined by the rule used in the dual simplex algorithm. Let D be the set of indices j for which $\pi_h^{r(T-1)} a_j^h < 0 \quad (r = r(T))$. Then choose $\phi_{s(T)}$ by

(11.8) $$\phi_{s(T)} = \min_{j \in D} \left\{ \frac{\pi_h^{p(T-1)} a_j^h}{- \pi_h^{r(T-1)} a_j^h} \right\} \geq 0 \qquad (r = r(T)) .$$

Note that

$$\phi_{s(T)} = \frac{d_s^{(T)}}{-\alpha_{s(T)}^r}$$

and (11.8) is the analogue in the dual problem of (8.3).

If the set D is void, then θ is at a maximum. If all $\gamma^i < 0$ in (11.3), then θ can be increased without bound. These are the only two automatic terminations in PLP. The following theorem is of interest.

<u>Theorem</u>: If the choice of $r(T)$ for (11.4) is unique and if D is not void, then there exists a finite range of θ, $\theta_{r(T)} \leq \theta \leq \theta_{r(T)}+\epsilon$, for which the solution obtained by replacing $\mu^{i(T-1)}_{r(T)}$ by $a^{i}_{s(T)}$, where $s(T)$ is determined by (11.8), is both feasible and optimal.

A proof can be found in Reference 4.

A separate control code for the computer is used in doing PLP. It always starts from a prior optimal, feasible solution. Due to (11.8), the iterations are longer than the regular code.

XII - MULTIPLE PHASES

It is somewhat difficult to parametrize the optimizing form since the analogue of γ^{i} would be a row vector of n+1 elements. As an alternative, provision is made for multiple optimizing forms which can be made to differ by finite amounts in any desired way. Of course, it is not contemplated that two such forms will be drastically different since that is equivalent to two different problems. In such a case, it would be better to start the second one from the beginning or from the end of Phase I.

It is also possible to split Phase I into multiple phases. This will not be discussed further since its application is limited and it generalizes easily from the discussions given.

It will be easier to describe the use of multiple phases if a specific example is used. Let it be required to optimize three forms and to start the problem with a Phase I. Then the auxiliary form must be

(12.1) $$a^{3}_{j} x^{j} + x^{n+1} = 0 \qquad (j = 3, 4, \ldots, n).$$

The form to be optimized first (after Phase I) must be

(12.2) $$x^{2} + a^{2}_{j} x^{j} = 0 \quad (\text{maximize } x^{2}).$$

Similarly, the other two forms to be optimized in turn must be

$$(12.3) \qquad x^1 \qquad + a^1_j\, x^j = 0 \quad (\text{maximize } x^1)$$

$$(12.4) \qquad x^0 \qquad + a^0_j\, x^j = 0 \quad (\text{maximize } x^0)\ .$$

The initial restraint equations will be

$$(12.5) \qquad a^i_j\, x^j + x^{n+i} = b^i \qquad (i = 4, 5, \ldots, m;\ \ j = 3, 4, \ldots, n).$$

Thus the initial value of q is specified as q = 4 (total number of phases) giving p = q-1 = 3 (number of " Phase II's"). The other two parameters required are z = 1 (number of " Phase I's") and $\zeta = 3$ (index of sum row.)

At the end of Phase I, p, q and z will all be reduced by 1 giving p =2, q = 3, z = 0 so that x^2 will now be maximized disregarding x^1 and x^0, (12.1) will now be considered a restraint equation the same as (12.5), and the phase will be recognized as a Phase II.*

When x^2 reaches a maximum, p will be reduced by 1 to p = 1 but q will remain at 3 so that x^1 will be maximized disregarding x^2 and x^0. Similarly, when x^1 reaches a maximum, p will be reduced to p = 0 with q still remaining at 3 so that x^0 will be maximized disregarding x^2 and x^1. In other words, p is reduced each phase but q is reduced each phase only as long as z can also be reduced. All three must remain non-negative, obviously.

---

* A Phase I is terminated when the variable being maximized reaches zero. A Phase II terminates when all $d_j \geq 0$. These criteria are not always equivalent even in Phase I. If b^1 is representable with fewer than m of the columns a^i_j (i, j > 0), then several artificial columns may remain in the basis at zero level at the end of Phase I. In this case, the d_j corresponding to the Phase I pricing vector will not all be non-negative. If Phase II starts with artificial columns in the basis (other than a^i_0 and a^i_{n+1}), then a^i_{n+1} may be eliminated in Phase II but one a^i_{n+k} will always remain.

<u>References.</u>

The following provide background material on the revised simplex method.

1. Dantzig, G. B., Alex Orden and Philip Wolfe, <u>Notes on Linear Programming, Part I The GENERALIZED SIMPLEX METHOD for MINIMIZING A LINEAR FORM UNDER LINEAR INEQUALITY RESTRAINTS</u>, RM-1264, The RAND Corp., Santa Monica, Calif.

2. ____________, ________, <u>Part II, DUALITY THEOREMS</u>, RM-1265, The RAND Corp.

3. Dantzig, G. B., Wm. Orchard-Hays, <u>Notes on Linear Programming, Part V, ALTERNATE ALGORITHM FOR THE REVISED SIMPLEX METHOD Using a Product Form for the Inverse</u>, RM-1268, The RAND Corp.

4. Orchard-Hays, Wm., <u>BACKGROUND, DEVELOPMENT and EXTENSIONS OF THE REVISED SIMPLEX METHOD</u>, RM-1433, The RAND Corp.

The following discuss the original simplex method.

5. Cowles Commission Monograph No. 13, <u>ACTIVITY ANALYSIS OF PRODUCTION AND ALLOCATION</u>, T. J. Koopmans, editor. New York, John Wiley and Sons, inc., 1951. (See especially Chapters XXI, XXII, XXIII)

6. Charnes, A. and W. W. Cooper and A. Henderson, <u>AN INTRODUCTION to LINEAR PROGRAMMING</u>, New York, John Wiley and Sons, Inc., 1953.

7. Eisemann, K., <u>LINEAR PROGRAMMING</u>, Quarterly of Applied Mathematics, Vol. XIII, No. 3, October 1955.

ORGANIZATION OF THE CODES

The data assembly code, as the name implies, is an assembly program and contains the usual conversion routines, etc. as well as several special subroutines designed for the particular conventions used in this system. A detailed description of its organization would serve no useful purpose here. Complete listings of the program are available for those who have a technical interest in the coding. The preparation of the inputs are described under INPUT and the output is discussed under OPERATION OF THE CODES. We will merely note here that data is processed either from cards or from tape 6 and that output is to tape 5 and punched cards (binary.) Since the entire program is too long for the available part of core storage, a part of the data assembly program is stored on drum 1 (automatically, during loading) and recalled to core storage when needed.

The concern here is with the [illegible] and organization of the main routines, as described [illegible].

Conceptually, the high-speed store (HSS) is divided into two main sections: (i) the programs, constants, parameters and temporary storage,

(ii) the data, both original and transformed.

Similarly, auxiliary storage is divided into two parts, one part as permanent storage for (i), the other as permanent storage for (ii). Thus the entire HSS is in a sense "temporary storage". In practice, a considerable part of section (i) of HSS remains intact throughout a run but can be restored from auxiliary storage (drums) if necessary. The advent of the extremely reliable core-storage, however, makes the need of restoring unlikely. When HSS reaches the sizes now contemplated (that is, 32,768 words of core storage), auxiliary storage for the code and for certain data will be unnecessary. The main use for auxiliary storage is for the a_j^i matrix and the $\eta_r^{i(t)}$ vectors. Magnetic tapes are used for both. On the IBM 704, three tapes are actually used for the $\eta_r^{i(t)}$ vectors as explained below.

The data section of HSS is divided into four regions:

the H-region for the basis headings: $m+1$ words

the V-region for the values β^i of the current solution, maintained in double-precision, floating point: $2(m+1)$ words

the W-region for work space in generating $a^i_{s(T)}$ and $\pi_i^{(T)}$ and other purposes: $2(m+1)$ words.

the T-region for temporarily holding the a^i_j matrix or $\eta_r^{i(t)}$ vectors or as much as possible of either: the remainder of section (ii) which should be at least $4(m+1)$ words.*

The size of these regions is a function of the number of restraints. Their origins are computed by the data assembly program. In PLP, V-region is used as a second W-region in pricing. A duplicate of H- and V-regions, as well as the original b^i, is kept in auxiliary storage. This allows re-starting after an error and checking a solution by computing and printing

$$\epsilon^i = \mu_h^{i(T)} \beta^{h(T)} - b^i.$$

For PLP, it is also necessary to keep c^i and γ^i in auxiliary storage.

The program section of HSS is divided into seven regions:

(a) Temporary storage called COMMON.

(b) The main control region, called the MCR.

(c) A sub-routine for doing double precision, floating point addition, called DPFADD.

(d) A similar sub-routine for multiplication, called DPFMUL.

(e) A routine called the distributor (DISTRB), explained below.

(f) Space for the largest sub-routine. The first location is called SRORIG,

---

* The origin of T-region has been arbitrarily set at 2048 for all problems.

(g) Universal constants, parameters, origins etc., called K-region. The need for COMMON, DPFADD, DPFMUL and K-region is obvious. Their locations are permanently fixed and may be referred to from any program in the system. All sub-routines are closed. The K-region contains certain cells whose contents are fixed only for part of an iteration or other sub-sequences of the problem. There are conventions on when and by what routine these are to be changed. COMMON is always available to any routine.

The function of the MCR is to make the major decisions and call for the proper sequence of operations. Most of the actual work is delegated to sub-routines. Besides the MCR for the main algorithm, there is one for PLP and one for re-inverting a basis. Other auxiliary programs are easily created by coding a new MCR.

The MCR calls for a major operation by linking to DISTRB with a pseudo-operation. DISTRB calls in the proper sub-routine from auxiliary storage and loads it into HSS starting at SRORIG. Control is then transferred to the sub-routine which returns control to the MCR after completing its function. If some other arrangement for linking to sub-routines is desired (as for example, when HSS is very large) it is only necessary to change DISTRB to arrange for the proper routine to take over in the proper way.

There are fourteen standard sub-routines. Others may be added for special purposes if desired, up to the limit of auxiliary storage to hold them. More importantly, if an improved or a special version of one is developed, it can replace the old one merely by exchanging the proper binary cards. These sub-routines, together with DPFADD, DPFMUL, K-region and a special loading routine, exist in binary cards which constitute a basic deck. The origins card produced by the data assembly is put ahead of, and the MCR behind this basic deck.

As soon as the special loading routine is in HSS, it takes control and loads the remainder of the basic deck and the MCR, storing them in auxiliary storage (drums). Since many of the sub-routines require addresses to be set which depend upon m, a loading interlude is performed to do this initialisation. This is accomplished by an initialising routine which goes with a sub-routine, the two being loaded simultaneously into HSS. Control is then sent to the initialising routine which does its job once and for all and returns control to the loader . The loader stores the initialised sub-routine in auxiliary storage and builds a catalogue of locations into DISTRB. The MCR takes over control when loading is complete.

The fourteen sub-routines (called "codes") are as follows. Some are used for multiple purposes which are controlled by internal switches.

Code 1. Load the binary cards produced by the data assembly and store in appropriate places. If the a_j^i cards are included, they are transferred to tape. Subject to a switch, a value may be loaded from a special card for f_p. By means of an internal switch and the use of control cards which it loads, this load routine is able to distinguish various cases, initial start, re-start, start of PLP, re-start of PLP, so that the proper information can be stored in auxiliary.

Code 2. Store the current solution in auxiliary (drums). The "current solution" is defined as K-, H-, and V-regions.

Code 3. (Normal) Form the row f_i in W-region, and record whether any variables are infeasible.

(During PLP) Form δ_i^p in V-region and δ_i^r in W-region.

Code 4. (Normal) Compute $\pi_h^{(T)} = f_i \, \pi_h^{i(T-1)}$ in W-region.

(During PLP) Compute $\pi_i^{p(T-1)}$ in V-region and $\pi_i^{r(T-1)}$ in W-region.

Code 5. (Normal) Price the matrix a_j^i and choose $d_s^{(T)}$.

(During PLP) Choose $\varphi_{s(T)}$ as described in Part A,* Sect. XI.

Subject to an external switch, recognize "Curtain" marks in a_j^i.

Subject to instructions from the MCR, make the following check:

(Normal) If $d_j^{(T)} \neq 0$, (during PLP) if $\pi_i^{r(T-1)}, a_j^i \neq 0$ for $j \neq j_{r(T)}^{(T-1)}$

then test for j occurring in $j_h^{(T-1)}$. If it does, an error has occurred. This check is very valuable for detecting errors before much more computing is done. It does consume some amount of time, however, which increases with n.

Code 6. Load $a_{s(T)}^i$ from a_j^i matrix into specified region as double-precision, floating point vector.

Code 7. Multiply $\pi_h^{i(T-1)} \; a_{s(T)}^h$ (or any vector in specified region).

Code 8. (Normal) Choose the index r(T) as discussed in Part A, Sect. VIII. For arbitrary transformations or inversion, choose r(T) by:

$$\left| \alpha_{s(T)}^r \right| = \max_{i \in A} \left| \alpha_{s(T)}^i \right|$$

where A is the set of indices i for which $j_i^{(T-1)} = 0$ (artificial).

Code 9. (Normal) Compute $\eta_r^{i(T)}$ from $\alpha_{s(T)}^i$ and transform $\beta^{i(T-1)}$ to $\beta^{i(T)}$.

(PLP, first entry) Compute $\beta^{i(T)}$ and $\theta_{r(T)}$ from $\beta^{i(T-1)}$, r(T) and $\gamma^{i(T-1)}$. (Part A, Sect. XI). Then compute $b^{i(T)} = b^{i(T-1)} + \theta_{r(T)} c^i$.

---

* "Part A" is used here to refer to DETAILED WRITE-UP.

(PLP, second entry) Compute $\eta_r^{i(T)}$ from $\dot{a}^i_{s(T)}$ and transform $\gamma^{i(T-1)}$ to $\gamma^{i(T)}$. (All these operations are very similar, even more than appears at first glance. The variations are merely switches.)

Code 10. Delete zeros from $\eta_r^{i(T)}$ and index non-zero elements. Store condensed vector in auxiliary storage (drums; cf. Code 13).

Code 11. Multiply out $\mu_h^{i(T)} \beta^{h(T)} - b^i = \epsilon^i$ taking $\mu_h^{i(T)}$ from original a^i_j matrix by referring to $j_h^{(T)}$. ϵ^i is left in T-region and b^i in W-region for printing.

Code 12. Print program. This program is quite elaborate and longer than the other codes. Special provisions for it need not be discussed here except to say that appropriate captions and identification of columns are printed for each type of print-out, of which there are 14. The print output can also be put on the fifth tape, if desired, for later printing. In all print-outs, there are 5 columns of which the first three are: $j_i^{(T)}$, $\beta^{i(T)}$ and $i = 0, \ldots, m$. The last two are the contents of W- and T-regions.

Code 13. This performs an "end-of-stage" in which some of the $\eta_r^{i(t)}$ are transferred from drum to tape. See below.

Code 14.-18. (Undefined , except for code 14 used with a special MTR.)

Code 19. Automatic restart program for recovering after an error by returning to the beginning of the iteration. Clearly, this is highly dependent on the particular machine. The important points to note are that all programs, the original data, the current solution at beginning of the iteration, and the $\eta_r^{i(T)}$ vectors must be intact somewhere in the machine. Also this code must have been previously instructed where to return control to and be able to restore all equipment (e.g. tapes) to the proper positions. It must, of course, be activated by some external means.

Besides the physical organization above, there are also dynamic groupings, several of which have already been discussed: the iteration, the phase, maximization of $\lambda<0$, arbitrary transformations, etc. Another grouping is called a <u>stage</u> and always consists of an integral number of iterations. It has nothing to do with phases or other mathematical aspects but is simply an operational device. It was devised for the IBM 701 and has been carried over to the 704, for slightly different reasons. Some analagous arrangement is probably necessary on any machine.

Most of the advantage of condensing the $\eta_r^{i(t)}$ vectors would be lost if a separate access to auxiliary storage had to be made for each one. It is desirable that as big a chunk of these transformation vectors as possible be recalled at one time. The limiting factor is the size of T-region. Hence as the $\eta_r^{i(t)}$ are generated, they are stored on a drum until one more would exceed the capacity of T-region. At this point an <u>end-of-stage</u> procedure is performed. Before describing this, it is necessary to explain the use of three tapes for the $\eta_r^{i(t)}$ vectors.

The tapes on the 704 can be back-spaced and additional records can be added to an existing file. However, they cannot be read in a backward direction and the back-space is somewhat slow. Hence the records are stored in reverse order on a second tape for use in computing $\pi_i^{(T)}$. The third tape is used alternately with the second from stage to stage. Though this costs some copying time, it does provide a spare (and checked) copy of the tape at

The end-of-stage procedure is as follows:

(1) Compute ϵ^i.

(2) Print $j_i^{(T)}$, $\beta^{i(T)}$, i, b^i, ϵ^i.

(3) Transfer the $\eta_r^{i(t)}$ on drum to an additional record on the forward tape.

(4) Transfer the $\eta_r^{i(t)}$ on drum to the first record on the free backward tape.

(5) Copy old backward tape to remainder of new backward tape.

(6) Punch out binary cards containing K-region, $j_i^{(T)}$ and $\beta^{i(T)}$. During PLP, also punch $b^{i(T)}$ and $\gamma^{i(T)}$.

(7) Adjust the necessary bookkeeping parameters.

An end-of-stage may be forced by an external switch. It also occurs at the end of Phase I and terminations. Steps (1) and (2) above may be forced without the others at the end of a cycle by an external switch. This would be desirable [illegible] above tape manipulations.

To restart from the end of a stage, it is only necessary to use the punch-outs to replace the corresponding original data cards and put the tapes back on the same units.

Similar simple hand collating of blocks of punch-outs with the original data cards (binary) and changing of the MCR are all that are necessary to set up the deck for PLP or re-inverting the basis. Special short MCR's are usually accumulated for such things as: transforming a new right hand side, computing and printing the d_j, altering a vector in the basis, etc. A special routine is provided for copying a $\eta_r^{i(t)}$ tape in reverse order so that a problem can be picked up after bad luck with tapes.

OPERATION OF THE CODES

DATA ASSEMBLY CODE

The data assembly program (which will be discussed only briefly) reads an identification card followed by a card with the 5 parameters. After this comes the basis headings, if any, FIRST B, the b^j, etc. as described under INPUT. Only non-zero elements are entered, with their proper indices.(Zero numbers will cause no difficulty, only a waste of time and space.) The program punches in binary cards:

The identification card (binary coded) which must be placed after the matrix transition card before loading by main code.

A self loading origins card needed by the loader for the main routines which must be placed on the front of the main code deck.

Two blank separators. (discard)

Initial X-, E-, and V-regions followed by a blank (retain.)

Right hand side for loading in W-region.

(Optional) Blank, auxiliary right hand side for loading in W-region.

(Optional) Blank, matrix for loading in T-region. Multiple records are separated by blanks (retain but not after last record.)

Matrix transition card (9 punch in col.37 only.)

For error detection, see list of stops.

ORDERING OF MAIN CODE DECK

The ordering of the cards is very important. If temporary correction cards are used, they must not be placed after the last card of a routine unless they load into the highest numbered cell used by the routine. The loader keeps track of first and last location of each routine except for MCR's, and uses this information for loading the routines on the drums and building the table for use by DIPERD. Following each routine, a title card identifies the code just loaded. The 9 row left is blank. The 9 row right contains the code number in the decrement. Sub-MCR's are identified by zero and MCR's by minus zero in 9 row right. The print program is loaded on drum 2 first as though it were a sub-MCR. However, this causes no difficulty since DISTRB does not recall the print program to HSS directly. A pseudo-code 12 (loaded in the normal way) recalls the real print program from the known location at the beginning of drum 2. The print program occupies the high part of T-region when in use. If another sub-MCR is loaded later in the loading sequence, DISTRB recalls this when Code 0 is called for by the MCR. DISTRB does not recognize "Code -0".

The binary cards are identified by Hollerith characters in cols. 73-80. The basic deck is as follows.

CLR REGS	1 card self-loading program to clear index registers and load next card. (Preceded by origins card punched by data assembly which leaves index regs. improperly for XIABSLDR.)
XIABSLDR	1 card self-loading load program for loading LPBASE.
LPBASE01 : LPBASE13	Several service routines: DRBOOT, a drum "bootstrap" for activating Code 19; DPFADD; DPFMUL; DISTRB with blank table; and XXECLD, the actual load program discussed above.
LPBASETR	Transition card to give control to XXECLD.
LPRINT01--LPRINT25	The actual print program.
LPRINTTL	Title card for print program.
LPCD0101--LPCD0105	Code 01
LPCD01TL	Title card

LPCD0201--LPCD0202, LPCD02TL   Code 02, etc.

LPCD0301--LPCD0305, LPCD03TL

LPCD0401--LPCD0410, LPCD04TL

LPCD0501--LPCD0511, LPCD05TL

LPCD0601--LPCD0603, LPCD06TL

LPCD0701--LPCD0706, LPCD07TL

LPCD0801--LPCD0806, LPCD08TL

LPCD0901--LPCD0909, LPCD09TL

LPCD1001--LPCD1004, LPCD10TL

LPCD1101--LPCD1109, LPCD11TL

LPCD1200	Pseudo-code 12 for calling print program into HSS.
LPCD12TL	Title card.

LPCD1301--LPCD1309, LPCD13TL

LPCD1901--LPCD1904, LPCD19TL

LPSINV01 ⋮ LPSINV10 LPSINVTL	Sub-MCR for inversion code. May be left in basic deck if desired, as long as no other sub-MCR is used in which case the capacity of the drums might be exceeded.

THE THREE MAIN MCR's

The basic MCR is for the composite simplex algorithm including Phase I, multiple phases and arbitrary transformations. The cards are identified by

LPCOMM01--LPCOMM16, LPCOMMTL.

These follow the basic deck described above.

A special MCR is used for PLP. The cards are identified by

LPPLPM01--LPPLPM10, LPPLPMTL.

When starting a run to do PLP, these cards are used instead of LPCOMM01, etc.

Another MCR is used to re-invert a given basis. Put LPSINV01, etc. at the end of the basic deck (see above) and use the following MCR cards.

LPINVM01--LPINVM15, LPINVMTL.

The sub-MCR operates first to organize the work. However, the operation of this is automatic. LPSINV01--LPSINV10, LPSINVTL, LPINVM01--LPINVM15, LPINVMTL can all be left together and considered as one MCR, if desired.

SWITCHING MCR's DURING A RUN

It is sometimes desirable to start with one MCR and then switch over to another one without getting off the machine. It is not necessary to use the punch-outs and reload the basic deck to do this. However, if a switch-over is to be made to the inversion MCR, the sub-MCR must have been loaded with the basic deck initially.

The special switch-over procedure uses Code 19, sense switch 6 and a special transition card. There are two of these, one for use with LPCOMM or LPINVM marked PIKUP TN. The other is for use with LPPLPM explained below.

To switch over to either LPCOMM or LPINVM, proceed as follows:

1. Put the following sequence of cards in the reader:
   - MCR without title card
   - PIKUP TN
   - Blank card
   - Matrix transition card (9 in col. 37)
   - Identification card punched by data assembly
   - Three blank cards
2. Put Switch 6 down.
3. Push CLEAR button.
4. Push LOAD DRUM button.
5. Mcr is read in and machine halts. Put Switch 6 up.
6. Push START button to proceed.

If it desired to read a scale factor card (f_p), then Switch 6 is left down in 5 and the f_p card is put after one of the blanks following the identification card.

To switch over to LPPLPM, the procedure is the same except for the cards used which are:

LPPLPM01--LPPLP10
The c^1 cards punched by the data assembly (which load to W-region)
PLP SPECL PIKUP TN (card is punched this way) and two blank cards.

An f_p card cannot be loaded for PLP in this manner.

LENGTH OF RUN

The codes have been designed to operate continuously until either

(a) a phase is complete,
(b) switch 1 is put down (arbitrary end-of-stage and halt), (also sw.6, see below.)
(c) the value of f_p must be changed, or
(d) an error is detected.

Whenever a stop occurs, hitting the START button will cause the program to do the "right thing", insofar as possible. This may be either

(a) to continue the calculations,
(b) to try again the sequence of operations in which an error was detected, or
(c) to lock-up and do nothing.

If the machine has performed everything required, it will lock-up with all four sense lights on. If the error is such that it cannot be corrected, the automatic drum restart (Code 19) may be used to repeat the iteration. This is activated by clearing MBS (CLEAR button) and pushing the LOAD DRUM button. Be sure switch 6 is up when doing this.

All stops are described in the list given later below. The ones normally to be expected are 0, 1, 6, 7, and 27. After certain stops (e.g. 5, 122) Switch 1 must be put down before hitting START (to get an end-of-stage) in order to stop the machine at the proper point. The job may be taken off the machine after any end-of-stage and restarted from this point. The punch-outs replace the corresponding original binary data cards, i.e. K-, H-, V-regions and, for FLP, W-region ($b^{i(T)}$). The binary data cards for COMMCR and FLPMCR are compared below.

ORDERING OF THE BINARY DATA CARDS

These cards follow immediately after the MCR title card.

LPCOMM		LPFLPM	
K-region	(Replace with punch-outs for restarting)	K-region	
H-region		H-region	
V-region		V-region	
		Blank card	insert to start FLP
		W-region ($b^{i(T)}$)	
		T-region ($\gamma^{i(T)}$) *	
- - - - - -	- - - - - -	- - - - - -	
Blank card → (to Blank card above)		Blank card	
W-region (b^i) → (to W-region above)		W-region (c^i)	
Blank card	(Optional)	Blank card	
a^i_j matrix (T-region)		a^i_j matrix (T-region)	
Matrix transition card		Matrix transition card	
Identification card		Identification card	
Blank card		Blank card	
f_p card	(Optional)**	f_p card	
2 blank cards		2 blank cards	

* and **, see top of next page

(Keyed from last page)

* Current $\gamma^{i(T)}$ vector does not exist initially. For starting PLP it is substituted for by a card which is blank except for 9-punch in some (any) column 38-72. This does not replace the blank following.

** Read only if Switch 6 down. The two words in 9-row are the value, the binary point being between columns 26 and 27. If f_p is to be negative, put a 9-punch in both columns 1 and 37.

To start an inversion run, use same data set-up as for LPCOMM with the punch-outs whose H-region specify the basis to be inverted. Everything is automatic provided the designations UP001, UP002, etc. have been used for legitimate unit vectors δ_1^i, δ_2^i, etc. If any $j_h < 0$ are left over from original set-up, they will be treated as zero and discarded.

PRINTED OUTPUT

The captions on the print-outs describe the reason for printing. The identification card is always printed first, followed by a line with iteration number (T), stage number less one (i.e. number stages completed), form number of maximand (i.e. current value of p), caption and, for some prints, "S IS ___" with appropriate vector name filled in. This means a_S^i just came into the basis or is to be multiplied by θ for unbounded solutions. The third line is a set of headings for the five columns of printing of which the first three are always:

J for $j_i^{(T)}$; BETA for $\beta^{i(T)}$; I for running index i.

The fourth column may be headed by any of the following:

B for b^i (or $b^{i(T)}$ in PLP)

E for $\eta_r^{i(T)}$

G for $\gamma^{i(T-1)}$

A for $a_{e(T)}^i$

The fifth column may be either

ER for ϵ^{i} (<u>error</u>) or

PI for $\pi_{i}^{(T)}$

or, in one instance, blank.

The fourteen captions with the corresponding headings for the fourth and fifth columns are as follows:

1.	CYCLE PRINT	S IS ___	B	PI	(forced by Switch 4)
2.	NEW SOLUTION	S IS ___	B	PI	(occurs every iteration in PLP unless $\theta_r = 0$.)
3.	CHECK SOLUTION		B	ER	(forced by Switch 6)
4.	END OF PHASE ONE		B	ER	
5.	END OF STAGE		B	ER	(drum 3 full or Switch 1 down)
6.	NO FEASIBLE SOLUTION		B	ER	(see stops 2 and 4 below)
7.	FEASIBLE SOLUTION		B	ER	(see stop 3 below)
8.	OPTIMAL SOLUTION		B	ER	(no stop until after 9)
9.	PRIMAL-DUAL SOLUTIONS		B	PI	(final print after 8, 13 or 14)
10.	UNBOUNDED SOLUTION	S IS ___	A	PI	(Terminal 3)
11.	RIGHT HAND SIDE OPEN	S IS ___	C	PI	(one of 2 possible PLP terminations, similar to 10)
12.	MATRIX SINGULAR	S IS ___	A	(blank)	(Inversion or arbitrary trans.)
13.	BASIS INVERTED		B	ER	(proper termination for inversion, no stop until after 9)
14.	THETA AT MAXIMUM		B	ER	(the other possible PLP termination, similar to 8; no stop until after 9)

<u>SENSE SWITCHES</u>

(if on)

1. Force end-of-stage and halt.
2. Omit the d_j check in Code 5.
3. Put all print output on tape 6, including any printed on-line, and do a cycle print (to tape) every iteration.
4. On-line cycle print.
5. Recognize "curtain" marks in a_j^i .
6. a. While loading or after stops 2 or 3: load value of f_p from card.
   b. While running: perform a check print at end of iteration and halt. (This is independent of cycle print and follows it.)
   c. When using Code 19 (automatic pick-up), load new MCR and halt.

ERROR STOPS AND OTHER HALTS

There are many error stops built into the codes. In order to facilitate de-bugging of problems, they have been left separate and distinct and identified with a number. Due to the way the codes are organized, it is not feasible to identify these stops by the HSS location. In any event, this would not permit easy reference. Instead, the stops are all "Halt and Proceed" orders with the stop number in the address. Hence when the machine stops, the identifying number appears in the address part of the Storage Register. All references to these stops are in octal.

Error Stops in Data Assembly Program

4444	Tape 6 (input) will not read properly after 5 tries.
5252	Zero overpunch on non-zero digit in a_j^i.
6666	$\lvert a_j^i \rvert \geq 8192$.
7700	$i > m$ for a_j^i.
7770	$i > m$ for b^i or c^i.
7777	$i > m$ for basis heading.
10421	Tape 5 (just written) will not read properly after 5 tries. This is only checked when the matrix is punched in cards.

Error Stops in Main Routines and Other Halts

These are systemized as follows.

Less than 100	- Legitimate stops in the MCR			
100 series	- Error stops in the MCR			
200 series	-	"	"	during pricing operations (Codes 4, 5)
300 series	-	"	"	in Code 7 (transformation of column)
400 series	-	"	"	in Code 9 (making change of basis, etc.)
500 series	-	"	"	in Code 11 (recomputing right hand side)
600 series	-	"	"	during loading.
700 series	-	"	"	from tape check failures.

The complete list is in the following table. The abbreviation e-o-s is used for end-of-stage and "S-x/START" for "Put Switch x down before hitting START button."

Octal Number	Reason for Stopping.	Result of START	Region of Stop
0	Switch 1 down, e-o-s finished.	Continue	MCR (any)
1	End of a Phase I, e-o-s finished.	Continue	COMM MCR
2	No feasible solution with present f_p; no e-o-s but f_p set to zero. S-6/START if a different value of f_p is to be loaded from card.	Continue with new f_p	COMM MCR
3	Feasible solution but $f_p = 0$; no e-o-s but f_p set to 1. S-6/START if a different value of f_p is to be loaded from card.	Continue with new f_p	COMM MCR
4	Absolutely no feasible solution (Terminal 1 or $f_p = 0$ with $\lambda \neq 0$); e-o-s done.	Nothing	COMM MCR
5	Solutions unbounded (Terminal 3), printing done. S-1/START.	Do an e-o-s	COMM MCR
6	Optimal solution for this phase, both printings and e-o-s done but more phases.	Continue	COMM MCR
7	Same as 6 but no more phases. (All 4 lights also on)	Nothing	COMM MCR
10	Check print completed. (Switch 6 down)	Continue	MCR (any)
15	Right Hand Side Open (in PLP), printing done. S-1/START.	Do an e-o-s	PLPM MCR
17	Theta at Maximum, both printings done and e-o-s finished. (All 4 lights also on)	Nothing	PLPM MCR
27	Inversion run all completed (printings, tapes, e-o-s done.(All 4 lights also on)	Nothing	INVM MCR
101	Arbitrary transformation called for but no negative j_h.(Clerical or machine err)	Nothing	COMM MCR
102	Specified matrix (arb. trnsfms.) singular, printing done. S-1/START.	Do an e-o-s	COMM MCR
103	No vector corresponding to negative j_h.	Nothing	COMM MCR
104	Monotonicity check on β^p failed.	Nothing	COMM MCR
120	On restart of inversion run, number of negative j_h disagrees with number of transformations yet to be done. (Clerical or machine error)	Nothing	SINV sub-MCR
121	End of file indication when trying to read record containing vector to be introduced during inversion. (Machine error)	Nothing	INVM MCR
122	Specified matrix (for inversion) singular, printing done. S-1/START.	Do an e-o-s	INVM MCR

Octal number	Reason for Stopping.	Result of START	Region of Stop
123	End of file indication when searching for vectors in basis to be inverted.	Nothing	SIDW sub-MCR
124	(Same as 121 but before proper tape record has been reached.)	Nothing	INVM MCR
125	Same unit vector given as legitimate and artificial.	Nothing	INVM MCR
126	Machine lost track of position of tape 5 during inversion. (Machine error)	Nothing	INVM MCR
240	Overflow in forming $x_1^{(T)}$ $(\geq 2^{25})$	Nothing	Code 4
247	Some $\left\|\eta_r^{1(t)}\right\| \geq 2^{25}$ in forming $x_1^{(T)}$	Nothing	Code 4
250	Overflow in computing $d_j^{(T)}$ $(\geq 2^{25})$	Nothing	Code 5
257	d_j check in Code 5 fails for some basis vector. Vector name in Accum. in BCD	Nothing	Code 5
310	DP ing point overflow on addition. $(\geq 2^{127})$	Nothing	Code 7
320	Same as 310 on multiplication.	Nothing	Code 7
400 series: 410 sub-series for overflow on DP floating addition		Nothing	Code 9
420 " " " " " " " multiplication		Nothing	Code 9
430 " " " underflow " " division		Nothing	Code 9

411 $\beta^1 + \beta^r \eta_r^1$ or $\gamma^1 + \gamma^r \eta_r^1$

413 $\beta^1 - \theta_r \gamma^1$ (FLP)

415 $b^1 + \theta_r c^1$ (FLP)

430 $-\frac{1}{a_s^r}$ (light 1 off)

$-\frac{1}{\gamma^r}$ (light 1 on)

420 $-\eta_r^r a_s^1$

421 $\beta^r \eta_r^1$ or $\gamma^r \eta_r^1$

422 $-\frac{1}{\gamma^r} \gamma^1$ (FLP)

423 $-\theta_r \gamma^1$ (FLP)

425 $\theta_r c^1$ (FLP)

Octal Number	Reason for Stopping.	Result of START	Region of Stop
510	DP floating point addition overflow in computing ϵ^1	Nothing	Code 11
520	Same as 510 for multiplication.	Nothing	Code 11
577	Cannot find all $\mu_h^{1(T)}$ in a_j^1 matrix.	Start Code 11 again	Code 11
600	Check sum error in loading code from cards.*	Continue ignoring error	EXECLD
607	Routine just loaded into HSS from cards is too long for remaining space on the drums.	Ignore this routine and continue loading.	EXECLD
610	Check sum error in data cards.	Continue ignoring error	Code 1

700 series, tape check stops. Whenever these stops occur, the record has already been tried (read) five times without success. Hitting START will cause the same record to be tried again, except as noted below. This will probably be of no avail save in exceptional cases.

Octal Number	Reason for Stopping.	Region of Stop
700	Advancing tape 2 ($\eta_r^{1(t)}$ forward) during inversion.	INVM MCR
701	Reading tape 3 (locations of basis vectors on tape 5) during inversion.	INVM MCR
702	Reading tape 2 to prepare backward tapes after inversion.	INVM MCR
703	Searching tape 5 for $a^1_{s(T)}$ for arbitrary transformation.	COMM MCR
705	Reading a_j^1 matrix from tape 5 during inversion.	INVM MCR
706	Reading a_j^1 matrix from tape 5 to prepare tape 3 for inversion.	SINV sub-MCR
724	Reading tape 3 or 4 in forming $x_i^{(T)}$ ($\eta_r^{1(t)}$ backward)	Code 4
725	Reading a_j^1 matrix from tape 5 in forming $d_j^{(T)}$ or loading $a^1_{s(T)}$.	Code 5
730	Reading tape 2 to compute $\alpha^1_{s(T)}$.	Code 7
750	Reading a_j^1 matrix from tape 5 in computing ϵ^1.	Code 11
760	Advancing tape 2 ready to write new record (end-of-stage)	Code 13

* A check sum error in the loading of LPBASE will cause a halt at 7776 octal with no special indication. Hit START to continue loading, ignoring error. The amount of error is in the Accumulator on all check sum stops. Using Code 19 to load a new MCR (special switch-over procedure), a check sum error in the cards causes a stop with no special indication. Should this occur, it would easily be overlooked.

The remainder of the tape check error stops occur in Code 13. Hitting START after these stops causes the program to continue as though there had been no error, at the operator's risk.

762 New record just written on tape 2 will not read correctly.
763 Newly written tape 3 may be no good (tape check on tape 4)
764 " " " 4 " " " " " " " 3)
765 762 and 763 combined.
766 762 and 764 combined.

There are two other error stops.

77777 Automatic restart from drum (Code 19) tried too early during inversion run. Reload deck.

717777 Code 19 got a check sum error in restoring HSS from drums. This check sum is carried on K-, N-, and V-regions only. The first 7 in this error number is in the tag field of the Storage Register.

RUNNING TIMES

It is impossible to give any realistic time estimates on a job. Time per iteration goes up with m, with n and with the number of iterations already performed. Elaborate formula have been worked out for this but they are not worth writing down although they have provided some comparative information. Nor is past experience a reliable guide. An inversion run on a 195 order system (m = 195) has been observed clipping along at 1 1/4 secs. per iteration. A 51 order system took 20 mins. to reach an optimal solution. Clearly these numbers do not bear a simple relation to one another. One of the critical factors is the density of the matrix. The following statements can be made, however.

1. The inversion MCR is the fastest in operation, at least twice as fast as the COMM MCR for a given problem.
2. The COMM MCR performs simplex cycles faster than any other known code considering the size of problems allowed, the accuracy maintained and the flexibility provided for.
3. FLP is invariably slower after a given number of iterations than the COMM MCR. It is probably wise to re-invert the optimal basis obtained by the COMM MCR before beginning extended calculations with FLP.