Stohr, Edward A.

Working Paper

A Mathematical Programming Generator System in APL

Discussion Paper, No. 348

Provided in Cooperation with:
Kellogg School of Management - Center for Mathematical Studies in Economics and Management Science, Northwestern University

Suggested Citation: Stohr, Edward A. (1979) : A Mathematical Programming Generator System in APL, Discussion Paper, No. 348, Northwestern University, Kellogg School of Management, Center for Mathematical Studies in Economics and Management Science, Evanston, IL

Discussion Paper No. 348

A MATHEMATICAL PROGRAMMING GENERATOR SYSTEM IN APL

by

Edward A. Stohr

Revised June 1979

A MATHEMATICAL PROGRAMMING GENERATOR SYSTEM IN APL

Edward A. Stohr
Associate Professor of Decision Sciences
Graduate School of Management
Northwestern University
Evanston, Illinois 60201

Abstract

This paper describes a mathematical programming generator which interprets problem statements written in the 'sigma notation' found in journal articles and textbooks. The syntax of the problem definition language is outlined and illustrative examples are given. The system has been implemented in APL. A unique feature is that the user can define objective function, constraint and right-hand-side coefficients as APL expressions. This leads to concise problem statements and also reduces data storage and processing requirements.

1. Introduction

The manual generation of the data for mathematical programming algorithms is a particularly tedious task which lends itself well to automation. In fact, for the large linear programs often found in practice, which may have thousands of constraints and variables, it is hard to imagine that this data could be generated by hand within a reasonable amount of time and with reasonable accuracy. There are two possible approaches to automating this task: (1) specially written programs which can read and validate the data for a particular problem structure and generate the required algorithm input in either 'tableau' or 'sparse matrix' form, or (2) mathematical programming 'generators' which can interpret any problem statement (not confined to a given structure) and are often combined with special data display and report writers for displaying the results of the algorithm [3], [5], [6]. This paper describes a mathematical programming generator written in APL which allows the user to use the capabilities and syntax of APL to help in problem definition.

The Mathematical Program Generator System (called MPGEN) accepts problem statements in the 'sigma' notation found in journal articles and textbooks. The problem statement is interpreted and the data for the tableau is generated as 'data triples' in the form (i,j,v) where i is the row index, j the column index and v the associated tableau value. The conventions adopted in generating the data triples are those used by the MPOS 'Multi-Purpose Optimization System' [2]. Thus the triples defining the right hand side constants have the value j=0, while the triples defining the objective function have the value i=0. The generated data triples can be used as input to: (1) Control Data Corporation's APEX (a large-scale mathematical programming system); (2) the MPOS system (a versatile system for small to medium-sized mathematical programming problems); or (3) mathematical programming algorithms coded in APL. The first alternative employs an option available under MPOS which has the capability of transcribing its input data into a format acceptable by APEX. The MPGEN system has been implemented in the APLUM [9], version of APL on Northwestern University's CDC 6600 computer.

The idea of using a 'sigma' notation appears also in [4], which contains a detailed language specification together with a description of many other components of their proposed system. The language described and illustrated in this paper has a quite different syntax. One of its chief features is that the variable coefficients, right-hand-side constants and variable indices can be specified as expressions in the APL host language. As the problem statement is interpreted, values are substituted for these expressions using the APL 'Execute' (sometimes called 'Evaluate') function, ⍎.[1] Using APL in this way helps to provide a concise problem statement and also reduces the need to preprocess the data. As a result data storage requirements can be greatly reduced.

The use of the sigma notation allows almost a direct transcription from the mathematical statement of the problem to the stored representation. The problem statements can be documented using 'comment' statements so that the problem definition can be readily understood; in fact, the problem statement can represent an abstract from the journal article, textbook or operation researcher's notebook in which the problem structure was originally defined. The problem statement is designed to 'look-like' a conventional mathematical programming statement in order to minimize the effort in transcribing from the mathematical statement to the input representation. The alternative of developing an APL-like syntax would represent a purer approach from the APL point-of-view but might be less acceptable to practicing management scientists and make the transcription less direct. The system is most useful in an educational environment since it allows one to quickly form a 'database' of linear and integer programming problems to illustrate applications in a wide range of areas such as cash management, capital budgeting, production planning and scheduling, transportation, facilities location, marketing and so on. In addition to the use of the system in defining mathematical programming problems, it can be used without change to generate systems of linear equations or inequalities.

This paper focuses on the problem definition language and especially those features of the implementation which rely on the capabilities of APL. The syntax of the language is described and illustrated in Section 2. The way in which APL can be utilized to form concise problem statements in a number of situations is described in Section 3. Section 4 briefly summarizes other features of the implementation including the provisions made for data entry and display, revision of the problem statements and storage of the output of the algorithms.

2. The Problem Definition Language

We first illustrate the use of the MPGEN system in the solution of a small linear programming problem:

$$\text{Maximize:} \quad 2x_1 + 3x_2 + x_3$$

$$\text{Subject to:} \quad x_1 + x_2 \leq 4$$

$$x_1 + x_2 + x_3 \leq 6$$

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

Using MPGEN an APL character matrix (say SAMPLE1) could be used to define this problem as shown in Figure 1:

```
      SAMPLE1
* SAMPLE PROBLEM 1 - ALGEBRAIC FORM
*
VAR=X(I), I IN ⍳3
*
MAXIMIZE
2X(1)+3X(2)+X(3)
*
X(1)+2X(2)≤4
*
X(1)+X(2)+X(3)≤6
* END
```

Figure 1
Algebraic Format

This is an algebraic form similar to that used by MPOS [2], and a number of other linear programming generators. Statements beginning with an asterisk are COMMENTs. The VAR= statement is a VARIABLE DECLARATION which is used by the system to assign columns to variables. The APL 'index generator' operator ⍳ is used to indicate that the INDEX SET for X is 1 2 3. Obviously, the above problem definition is only useful for the particular problem data shown. More generally the standard form of a linear programming problem:

$$\text{Maximize:} \quad \sum_{j=1}^{N} c_j x_j$$

$$\text{Subject to:} \quad \sum_{j=1}^{N} a_{ij} x_j \leq b_i, \quad i=1,2,\ldots,M$$

can be defined in MPGEN using 'sigma' notation as shown in Figure 2.

Problem Statement

```
                    SAMPLE2←DEFPROBLEM
COMMENT LINE        * SAMPLE PROBLEM 2 - SIGMA NOTATION
                    *
DATA DECLARATION    DATA=M,N,A(M×N),B(M),C(N)
                    *
VARIABLE DECLARATION VAR=X(J), J IN ιN                  ┌─ DECISION VARIABLE
                    *
                    MAXIMIZE
OBJECTIVE DEFINITION S C[J]X(J)                         ┌─ VARIABLE INDEX EXPRESSION
SUMMATION INDEX LINE J IN ιN                            ├─ INDEX SET EXPRESSION
                    *
FOR INDEX LINE      FOR I IN ιM
CONSTRAINT DEFINITION S A[I;J]X(J)≤B[I]                 ┌─ RHS COEFFICIENT EXPRESSION
                    J IN ιN                             ├─ VARIABLE COEFFICIENT EXPRESSION
                    * END
```

Figure 2

'Sigma' Notation Format and Definition of Terms

Here DEFPROBLEM is an APL function which allows the user to input a character matrix line-by-line; when the last line has been typed the user simply presses carriage-return on the next line to end the input. The APL variable SAMPLE2 then contains the defined character matrix. Modification and correction of the problem statement is performed using an EDIT function which invokes the APLUM Function Editor.

The DATA= line is a DATA DECLARATION statement (optional). It is used by the system to check that the required data (variables M,N,A,B,C) are present in the APL workspace and that the variables (A,B,C) have the indicated dimensions. If either of these conditions are false, the system will prompt the user either to input the required data as each line of the tableau is generated or to halt the problem interpretation.

An 'S' followed by a blank in an OBJECTIVE or CONSTRAINT DEFINITION line represents the algebraic symbol, 'Σ'. The range for the summation is indicated in the following SUMMATION INDEX LINE. In Figure 2 we have: J in ⍳N, meaning that J takes values in the SET 1,2,...,N. The rows for which a CONSTRAINT DEFINITION is defined are given by one or more FOR INDEX LINE's. In the example there is one such line which specifies that there are M constraint rows--I belongs to the INDEX SET 1,2,...,M.

The problem definition in Figure 2 obviously applies to <u>any</u> linear program; it is only necessary to define the appropriate cost and right-hand-side (RHS) coefficient vectors and the constraint coefficient matrix A. Thus the 'sigma' notation has two great advantages: (1) it is a more compact notation which corresponds almost exactly to the format used in the statements of Operations Research models,

(2) the problem definition is general in the sense that it is independent of the dimensions of the problem (number of variables and constraints involved). The disadvantage of the standard format for a linear program shown in Figure 2 is that it ignores any special structure which might apply to a particular class of problem. Thus, the user is required to construct the matrix A from the constraint equations of the problem. This is a laborious task and also requires the input and storage of unnecessary data since this matrix is generally quite sparse in practice.

Having defined the problem as above, the user can then run it using an APEX, MPOS or APL linear programming algorithm as illustrated for a more complex problem in Figure 4.

Before proceeding to a more comprehensive example we outline the rules for defining a linear (or integer) programming problem using MPGEN. A more complete description is given in [7].

The INDEX SET, VARIABLE INDEX, RHS and VARIABLE COEFFICIENT EXPRESSIONS (see Figure 2) can be defined by the user using the APL language. These expressions are evaluated when the problem is interpreted using the APL 'Execute' function. Imbedded in the expressions will be the data variables and constants which define a particular instance of the problem. With the exception of certain reserved names, the user can employ any valid APL variable names for the APL data variables, INDEX VARIABLES and DECISION VARIABLES.

Note that the indices for the linear program DECISION VARIABLES x_1, x_2, x_3 in the above were enclosed in parentheses in the problem statement--X(1),X(2),X(3) in Figure 1 and X(J) in Figure 2. The MPGEN

System uses the names in the VARIABLE DECLARATIONS plus the parentheses to recognize DECISION VARIABLES when interpreting CONSTRAINT and OBJECTIVE DEFINITIONS. In the current implementation a DECISION VARIABLE can be indexed by up to five indices separated by commas. Each INDEX can be a constant, a variable appearing in a SUMMATION INDEX or FOR INDEX LINE, or any non-parenthesized APL expression which returns a scalar result. During interpretation the MPGEN system evaluates each index in the VARIABLE INDEX expression separately using the APL 'Execute' function.

As will be illustrated later, allowing the user to employ APL statements within the problem definition helps in forming concise problem statements and reduces both data storage requirements and the need for preprocessing data. The danger in allowing this freedom, of course, is that error detection may become more difficult because the MPGEN system does not check the syntax of the APL expressions--this is done by the APL processor during the evaluation. This potential drawback is mitigated, if not eliminated, however by (1) 'trapping' any such error and providing an error message which displays the expression where the error occurred, and (2) by checking the presence and dimensions of all required data using the DATA DECLARATION statement.

OBJECTIVE and CONSTRAINT DEFINITION lines differ only in that the latter must contain one of the relational operators $\leqslant$, =, or $\geqslant$ together with a RHS COEFFICIENT EXPRESSION. The OBJECTIVE and CONSTRAINT DEFINITIONs consist of one or more VARIABLE TERMS separated by '+' or '-' operators A VARIABLE TERM has the following form

[One or more Summation Symbols] [Coefficient Expression]

Varname (Variable Index Expression)

where the square brackets indicate optional components. For example, the second constraint in Figure 1 has two VARIABLE TERMS - X(1) and 2X(2) while the CONSTRAINT DEFINITION in Figure 2 has only one - S C[J] X(J). Note that the summation symbols 'S' apply to only one variable; if more than one variable name appears in an OBJECTIVE or CONSTRAINT DEFINITION line, then each must have its own summation symbols.

If an expression contains one or more summation signs, it must be followed by a SUMMATION INDEX LINE in which the corresponding index variables and the values they are to assume are defined by one or more INDEX TERMS separated by commas. An INDEX TERM has the form:

Index variable name IN Set expression.

The SET EXPRESSION may be any APL expression which returns a positive scalar or vector result, e.g. ⍳N, in (Figure 2) or I[K;] in Figure 4). The result of the SET EXPRESSION defines the values taken on by the INDEX variable during the summation. The correspondence between summation signs and INDEX TERMS is obtained from the order (from left-to-right) in which the latter appear in the SUMMATION INDEX LINE. Note that the desired result of a SET EXPRESSION may depend on the value of a previously defined index. Thus, in Figure 3, we have K IN K, I IN I[K;] where K is an APL vector of index values for K and I is an APL matrix in which the rth row contains the index values for I when K=r. Since trailing zeroes in an index vector are ignored by MPGEN it is possible for the number of values assumed by the I index to vary with the value of K. Other facilities for defining SET EXPRESSIONS are defined in [7].

OBJECTIVE and CONSTRAINT DEFINITION lines and SUMMATION INDEX LINES can be continued if necessary on a succeeding line prefaced by a colon, ':', as shown in Figure 3 below.

The rows for which a CONSTRAINT DEFINITION applies are defined by one or more preceding FOR INDEX LINEs. These consist of the word 'FOR' followed by an INDEX TERM as defined above. Again, the SET EXPRESSIONS can be dependent on previously defined indices.

VARIABLE DECLARATION LINES also contain INDEX TERMS - one for each index variable (see Figure 3). The associated INDEX SETS must specify the full range of values which the index can take on in the problem statement.

An OBJECTIVE GROUP consists of a line containing the word MAXIMIZE (or MINIMIZE), the OBJECTIVE DEFINITION line, and (if the latter contains one or more summation symbols) the associated SUMMATION INDEX LINE. A CONSTRAINT GROUP consists of one or more FOR INDEX LINEs, the CONSTRAINT DEFINITION line, and, if required, a SUMMATION INDEX LINE. A GROUP and its associated data triples form a basic unit of data in the MPGEN system. All GROUPS are preceded and followed by one or more COMMENT lines. The user is free to write any desired descriptive material in the comment lines. If the user requests a print-out of the tableau after the problem statement has been interpreted the comment lines preceding each GROUP can be used as sub-headings for the report (the indexed variable names form the individual column headings).

In addition to the statements which define the mathematical program itself the user may insert other commands in the problem statement by the use of the EXECUTE statement. This has the form:

EXEC: valid APL expression.

The EXECUTE statement can be used, for example, to open, read and close files, to erase data variables which are no longer required and to perform conditional operations on the problem data.

The MPGEN system interprets the problem statement in one pass in line-by-line order. As the data triples defining each row of the tableau are generated they are written as records on a sequential 'coded' APLUM file. Thus core requirements are kept to a minimum and the system has the capability of generating quite large mathematical programming tableaux. In addition, to take advantage of the efficiencies to be obtained from linear programming codes such as MPOS which utilize a variant of the simplex method for variables with upper and lower bounds, such constraints can be automatically recognized by the MPGEN system. Modified data triples in the format required by MPOS [2] are then produced. The file of data triples and the APEX or MPOS 'Problem Statements' (if any) are normal CDC SCOPE Operating System files which can, for example, be read by FORTRAN programs. Together they contain all the information necessary to run the mathematical programming algorithms. If required, these files can be saved for later use in sensitivity analysis. An APL function, MODIFYDATA, can be used to retrieve and change individual data triples and to rerun the algorithm. Another function, MODIFYGROUP, allows the user to selectively reinterpret parts of the problem definition and to store the results in the file of data triples.

3. The Use of the APL EXECUTE Function in Defining Mathematical Programming Problems

The problem definition language described in the previous section is now illustrated using a more complicated example.

The sample problem, reproduced here in its original form, is taken from [1]:[2]

(20a)

$$\max_{x_{ijk\ell}, y_{ik\ell}} \sum_{k\in K} N_k \Bigg[\sum_{\ell\in L} \sum_{j\in J_\ell} \sum_{i\in I_k} v_{ijk\ell} x_{ijk\ell} - \sum_{\ell\in L} \sum_{j\in J_\ell} \sum_{i\in I_k} \Big(\sum_{r\in R_{j\ell}} p_{rj\ell} h_{rj\ell ik} \Big) x_{ijk\ell} - \sum_{\ell\in L} \sum_{i\in I_k^p} \Big(\sum_{s\in S_\ell} q_{s\ell} g_{s\ell ik} \Big) y_{ik\ell} \Bigg]$$

subject to:

(20b)

$$\sum_{k\in K} N_k \sum_{i\in I_k} m_{ik} x_{ijk\ell} \le M_{j\ell} ,$$

$$\forall\ j \in J_\ell,\ \ell \in L$$

(20c)

$$\sum_{k\in K} N_k \sum_{i\in I_k^p} t_{ik\ell} y_{ik\ell} \le T_\ell ,$$

$$\ell \in L$$

(20d)

$$\sum_{u\in U_k(i)} \sum_{j\in J_\ell} x_{ujk\ell} - |U_k(i)| y_{ik\ell} = 0 ,$$

$$\forall\ i \in I_k^p,\ \ell \in L,\ k \in K$$

(20e)

$$\sum_{\ell\in L} \sum_{j\in J_\ell} x_{ijk\ell} = 1 ,$$

$$\forall\ i \in I_k,\ k \in K$$

(20f)

$$\sum_{\ell\in L} y_{ik\ell} = 1 ,$$

$$\forall\ i \in I_k^p,\ k \in K$$

The above is actually a sub-problem of a 'system' pricing problem in which competing computer centers, ℓ, attempt to

choose optimal prices, $P_{rj\ell}$ and $q_{s\ell}$, for various computing resources such as CPU time and space on auxiliary storage devices. The MPGEN problem definition is stored in the character matrix USERPROBLEM as shown in Figure 3. In translating from the algebraic statement given above, variables represented by small letters in the original are represented by capital letters and variables represented by capital letters are represented by 'understruck' APL characters.

To form the objective function for MPGEN, the two terms in x in (20a) are combined to form a single VARIABLE TERM. The coefficient

$$(N_k[v_{ijk\ell} - \sum_{r\in R_\ell} P_{rj\ell} h_{rj\ell ik}])$$

is represented by the APL statement

```
(N[K] × V[I;J;K;L] - P[;J;L]+·× H[;J;L;I;K])
```

which utilizes the 'inner product operator +·× to perform the multiplication and summation over r. The VARIABLE COEFFICIENT EXPRESSION for y is formed in a similar fashion. In most current LP Generators the expressions for these coefficients would have to be evaluated and stored in a prior processing run. Allowing these operations to be evaluated by the APL host language at the time the tableau is generated eliminates both this prior processing step and the necessity to store the computed data.

Constraint (20d) also illustrates the use of an APL expression. Here, the vector U[K;I;] represents $U_k(i)$ (the set of data sets required by program i in 'usage class', k). Noting again that the system will ignore trailing zeroes in an index set, we obtain the cardinality of $U_k(i)$ using the APL expression +/0<U[K;I;].

```
      USERPROBLEM
* ////////////////////PRICING PROBLEM/////////////////////////// *
*
* USER PROBLEM
* ------------
* ASSIGNMENT OF DATA SETS AND PROGRAMS TO MINIMIZE COSTS
* FOR GIVEN PRICES SET BY COMPUTER SYSTEMS.
*
* MATH CENTER WORKING PAPER NO 168
* DEFINED AUGUST 2, 1978    E. A. STOHR
*
* ..... DATA DECLARATIONS .....
*
* DIMENSIONS
DATA=LN,KN,JN,IN,IPN,UN,SN,RN
*
* INDEX SETS
DATA=L(LN),K(KN),J(LN×JN),I(KN×IN),IP(KN×IPN),U(KN×IPN×IN)
*
* USER PROGRAM AND DATASET DATA
DATA=N(KN),G(SN×LN×IN×KN),H(RN×JN×LN×IN×KN)
DATA=V(IN×JN×KN×LN),M(IN×KN),T(IPN×KN×LN)
*
* SYSTEM PRICE AND CAPACITY DATA
DATA=P(RN×JN×LN),Q(SN×LN),M(JN×LN),T(LN)
*
* ..... VARIABLE DECLARATIONS .....
*
* X(I,J,K,L)=1 IF DATASET I OF USER K IS STORED ON DEVICE
* J OF SYSTEM L, 0 ELSE
VAR=X(I,J,K,L), L IN L, K IN K, J IN J[L;], I IN I[K;]
*
* Y(I,K,L)=1 IF PROGRAM I OF USER K IS RUN ON SYSTEM L, 0 ELSE
VAR=Y(I,K,L), L IN L, K IN K, I IN IP[K;]
*
* ..... OBJECTIVE FUNCTION .....
*
* MINIMIZE COST OF STORING DATASETS AND RUNNING PROGRAMS
* EQUATION (20A)
*
MINIMIZE
S S S S (N[K]×V[I;J;K;L]-P[;J;L]+.×H[;J;L;I;K])X(I,J,K,L)
L IN L, K IN K, J IN J[L;], I IN I[K;]
:-S S S (N[K]×Q[;L]+×G[;L;I;K])Y(I,K,L)
:,L IN L, K IN K, I IN IP[K;]
*
* ..... CONSTRAINTS .....
*
```

Figure 3
Definition of Pricing Problem

```
* EQUATION (20B) - SIZE CAPACITY CONSTRAINT
FOR L IN L
FOR J IN J[L;]
S S (N[K]×M[I;K])X(I,J,K,L)≤M[J;L]
K IN K, I IN I[K;]
*
* EQUATION (20C) - TIME CAPACITY CONSTRAINTS
FOR L IN L
S S (N[K]×T[I;K;L])Y(I,K,L)≤T[L]
K IN K, I IN IP[K;]
*
* EQUATION (20D) - DATASETS AND PROGRAMS ON SAME SYSTEM
FOR L IN L
FOR K IN K
FOR I IN IP[K;]
S S X(U,J,K,L) - (+/0<U[K;I;])Y(I,K,L)=0
U IN U[K;I;], J IN J[L;]
*
* EQUATION (20E) - DATASETS MUST BE ASSIGNED TO ONLY ONE DEVICE
FOR K IN K
FOR I IN I[K;]
S S X(I,J,K,L)=1
L IN L, J IN J[L;]
*
* EQUATION (20F) - REQUIREMENT THAT ALL PROGRAMS ARE RUN
FOR K IN K
FOR I IN IP[K;]
S     Y(I,K,L) = 1
L IN L
*
* ////////////////// END OF PROBLEM ////////////////////////////*
```

Figure 3
(continued)

Figure 4 illustrates the user interaction when running this problem. Note that four problems have previously been stored in the 'Library' and that the 'USER PROBLEM' described above has been given the identifier, 'PRICE'. The required data is already present in the workspace as verified by the DATACHECK listing. Figure 4 also shows the retrieval of another problem 'SAMPLE 1' from the library during the same terminal session. The problem statement is simply displayed ready for editing. However, the great reduction in human data preparation activities argues overwhelmingly for the use of the system - especially for large problems.

There are many other situations in which the power of the APL language can be utilized to help develop a concise problem statement. A few more examples are listed below:

(1) Logical Conditions: Suppose that the index variable I appears in one of the FOR INDEX lines for a CONSTRAINT GROUP and that the RHS coefficient should equal zero when I=1 and should equal one otherwise. Instead of generating a data vector for the RHS of zeroes and ones, the RHS COEFFICIENT EXPRESSION can be stated as $I \neq 1$.

As another example consider the constraint set:

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_{ij} \leq b_i \quad , \quad 1 \leq i \leq n$$

Since the INDEX SET EXPRESSION for j can depend on the value of i, these constraints can be handled in MPGEN by letting the index set for j be an (n X n-1) matrix with the appropriate coefficients for j in the ith row or (preferably) by the APL expression:

$$(\sim(\iota N) \in I)/\iota N$$

(2) Direct access to data via Random Access Files: Suppose constraint coefficients, a_{ij}, are updated periodically and stored on an APL random access file with 'file-tie number', FN. If the user-defined function POS I,J returns the relative position of a_{ij} in the file, the constraints in Figure 2 can be represented by:

```
FOR I IN ⍳M
S (FREAD FN, POS I,J) X(J) < B[I]
J IN ⍳N
```

This avoids the necessity to store data in the workspace and gives MPGEN the capability of generating extremely large tableaux. However, accessing data items individually in this way is time-consuming. A compromise solution is to use the EXECUTE statement to insert data retrieval (and erasure) commands into the problem definition before (and after) OBJECTIVE AND CONSTRAINT GROUPS. In this way, input-output costs can be reduced because larger amounts of information can be moved to and from direct access storage.

(3) User Defined Functions and Other APL functions: As illustrated above, any APL statement which returns a scalar can be inserted in a COEFFICIENT EXPRESSION. These statements can be regarded as real-valued functions of the index variables which define the row and column position of the coefficient in the tableau. The APL functions that might conceivably be used include the 'matrix divide' function which computes regression coefficients.

(4) Interactive Input of Data:[3] If objective function, constraint or right-hand side coefficient expressions vary from run to run (perhaps for sensitivity testing purposes) this data may be input interactively during problem interpretation using the MPGEN 'ASK' function. Thus the constraint in Figure 2 could be replaced by S (ASK) X(J) < B[I]. As this constraint is interpreted the user will be prompted to input the A[I;J] coefficients from the keyboard.

```
      RUNPROBLEM

OPTION? (OR TYPE 'HELP')
HELP
TYPE:
LIBRARY        - TO DISPLAY LIST OF PROBLEMS IN LIBRARY
GET (PROBNAME) - TO START WORK ON PROBLEM (PROBNAME)
PROBLEM        - TO DISPLAY/MODIFY PROBLEM STATEMENT
INTERPRET      - TO INTERPRET PROBLEM DEFINITION
TRIPLES        - TO DISPLAY/MODIFY DATA TRIPLES
TABLEAU        - TO DISPLAY PROBLEM TABLEAU
SAVE           - TO SAVE PROBLEM STATEMENT/TRIPLES/RESULTS
RUN            - TO RUN PROBLEM
MODE           - TO CHANGE MODE (APEX,MPOS, OR APL)
SET CONTROLS   - TO CHANGE REPORT AND LP CONTROL VARIABLES
STOP           - TO STOP

OPTION? (OR TYPE 'HELP')
LIBRARY
PRICE
SAMP1
SAMP2
SAMP21

OPTION? (OR TYPE 'HELP')
GET PRICE

OPTION? (OR TYPE 'HELP')
INTERPRET

--- LIST OF INTERPRETED STATEMENTS ---
DATA CHECK  LN,KN,JN,IN,IPN,UN,SN,RN
DATA CHECK  L(LN),K(KN),J(LN×JN),I(KN×IN),IP(KN×IPN),U(KN×IPN×IN)
DATA CHECK  N(KN),G(SN×LN×IN×KN),H(RN×JN×LN×IN×KN)
DATA CHECK  V(IN×JN×KN×LN),M(IN×KN),T(IPN×KN×LN)
DATA CHECK  P(RN×JN×LN),Q(SN×LN),M(JN×LN),T(LN)
S S S S (N[K]×V[I;J;K;L]-P[;J;L]+.×H[;J;L;I;K])X(I,J,K,L)-S S S (N[K]×+/Q[;L]×G[
      ;L;I;K])Y(I,K,L)
S S (N[K]×M[I;K])X(I,J,K,L)≤M[J;L]
S S (N[K]×T[I;K;L])Y(I,K,L)≤T[L]
S S X(U,J,K,L)-(+/0<U[K;I;])Y(I,K,L)=0
S S X(I,J,K,L)=1
S Y(I,K,L)=1
--- END OF PROBLEM INTERPRETATION ---

APEX, MPOS, APL OR STOP?
APL
TITLE FOR REPORT?
PRICING PROBLEM
---------------
```

Figure 4
Running the Computer Pricing Problem

```
OPTION? (OR TYPE 'HELP')
TABLEAU
5 /29/1979  11.30.48                                              PAGE 1

PRICING PROBLEM
---------------

          RHS  X1111  X2111  X1211  X2211  X1121  X2121  X1221  X2221  X1112
-------------------------------------------------------------------------------
0    ≤      0     ¯1     ¯9    ¯42    ¯14     ¯2    ¯52    ¯34    ¯16     ¯6
1    ≤    150      1      3      0      0      4      8      0      0      0
2    ≤     30      0      0      1      3      0      0      4      8      0
3    ≤     90      0      0      0      0      0      0      0      0      1
4    ≤    150      0      0      0      0      0      0      0      0      0
     :      :      :      :      :      :      :      :      :      :      :
```

```
OPTION? (OR TYPE 'HELP')
GET SAMP1

OPTION? (OR TYPE 'HELP')
PROBLEM

[12]  [□]∇
      ∇Z4567890123456789012Z4567890123Z4567
[1]    * SAMPLE PROBLEM 1 - ALGEBRAIC FORM
[2]    *
[3]    VAR=X(I), I IN ⍳3
[4]    *
[5]    MAXIMIZE
[6]    2X(1)+3X(2)+X(3)
[7]    *
[8]    X(1)+2X(2)<4
[9]    *
[10]   X(1)+X(2)+X(3)<6
[11]   * END
      ∇

OPTION? (OR TYPE 'HELP')
STOP
```

Figure 4 (continued)

4. Other Features

The problem definition language described in this paper provides a convenient and concise means for defining linear and integer programming problems. Because of its labor-saving characteristics, it should allow the operations researcher to implement models more easily and to experiment with alternative formulations. However, to provide a complete 'decision support system', many other facilities must be supplied, including: (1) routines to aid in data entry and to manage the processing of primary data to the form required by the model, (2) report generation routines which provide comprehensive and readable summaries of results and sensitivity analyses and (3) a system to help the user manage the exploration of a multitude of alternative models and assumptions and to maintain the links between the results of different runs and their data inputs. Data base management system techniques involving the use of a 'network' data model as described in [8] will be used in conjunction with MPGEN to achieve these objectives.

Footnotes

[1]The Execute Operator executes APL instructions written as a character string.

[2]The equation numbers correspond to those given in [1].

[3]I am indebted to Dr. Melvyn H. Schwartz of Vogelback Computing Center for this suggestion.

References

1. Balachandran, V. and Edward A. Stohr, "Optimal Pricing of Computer Resources in a Competitive Environment," Working Paper No. 268, Center for Mathematical Studies in Economics and Management Science, Northwestern University, 1978

2. Cohen, Claude and Jack Stein, "Multi-Purpose Optimization System, User's Guide, Version 3," Vogelback Computing System, Northwestern University, 1976.

3. Control Data Corporation, APEX II Reference Manual, Publication No. 59158100, 1974.

4. Fourer, Robert and Michael J. Harrison "A Modern Approach to Computer Systems for Linear Programming," Working Paper 988-78, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts, March 1978.

5. Haverly Systems, Inc., LP 360/370 Linear Programming System: User and Operating Manual, Sixth Edition, November 1977.

6. IBM Corporation, IBM Mathematical Programming System Extended (MPSX/370) Program Reference Manual, No. SH19-1094, 1976.

7. Stohr, Edward A., "MPGEN User Manual," Graduate School of Management, Northwestern University, 1978.

8. Tanniru, Mohan, "The Design of a Decision Support System," Ph.D. Dissertation, Graduate School of Management, Northwestern University, 1978.

9. Wiedmann, Clark, "APLUM Reference Manual," University of Massachusetts Computing Center, Amherst, Massachusetts, 1975.