Matrix Generation Using SAS® Software
Marc-david Cohen
SAS Institute Inc.
Cary, N.C.

Introduction

Linear programming has been a modeling tool of practical importance in many industrial and business settings. The technique enables analysts to study systems in an effort to improve performance and increase efficiencies. Production, transportation, and scheduling are three areas that have been modeled extensively using linear programming. Management Science, Interfaces, and Computers & Operations Research are examples of journals that have published numerous articles showing applications.

Analyzing systems using linear programming models usually involve several types of tasks. These include: mathematical model building, data collection, data keying, model solution, and solution reporting. Often the most time consuming work is associated with data collection and the data handling tasks involved in building, solving, and reporting. Although the actual time the computer spends in model optimization can be significant the vast majority of real time is spent on model building, solution reporting, and other data handling tasks.

A 'Matrix generator' is a term used for a language that simplifies the tasks associated with model building and data handling. PDS/MAGEN, GAMMA, MGG, OMNI, and GAMS are examples of several commercially available matrix generators. In this paper, we present a system of SAS macros, called MATGEN, that emulate a matrix generation language using the SAS language. We discuss the use of the macros, examine how they are integrated, and show several examples of their application. We also consider the advantages and disadvantages of using a macro approach to matrix generation.

The MATGEN language that we present is not intended to be an all inclusive matrix generation language. The purpose of this exercise is to provide a framework for developing a more complete language. As a result, the MATGEN architecture is designed for realistic size models and its limitations make it easy to describe without sacrificing its ability to be extended.

Components of a Decision Support System

In an industrial or business environment, problems in operations relating to efficient management periodically come to attention. If such a problem is persistent or in an area that has potential for significant savings, then resources are allocated toward problem solution. Problem study can be simplified by considering it within the structure of a problem solving paradigm. Figure 1 shows one such paradigm.

- Enumerate a specific problem
- Abstract from "real world"
- Formulate a mathematical model
- Measure model parameters
- Build model
- Solve for decision variables
- Implement solution

Figure 1. A paradigm for problem solving.

Once a general problem is stated it must be narrowed to a manageable size. For example, in the general framework one may have identified the manufacture of a cereal as an area were efficiency can be gained. On closer examination, identifying the least cost mix of several grains that meets the nutritional needs of the cereal is of particular interest. This reduction from a general question of manufacture to the more specific one of particular product mix not only further clarifies the problem, but also helps identify those variables which control the process. These variables are called the decision variables. Finding values for them is the goal of the analyst.

With the problem more clearly defined, the process can be studied for those features that significantly affect the decision variables. A model can be abstracted from these observations. Knowledge and understanding about the process is paramount at this stage. The abstraction of a model from the "real world" not only requires experience with the particulars of the process under study but also with the tools of quantitative mathematics.

The abstracted model is then formulated as a mathematical representation of the process. If done carefully, the formulation captures the significant details of the processes in a way that faithfully represents the structure of the process and allows one to solve for the decision variables.

Next, the parameters of the model must be measured. While the formulation of the model is the representation of the structure of the process, the model parameters are the data associated with the particular realization in hand. For example, the structure of the product mix problem mentioned above is independent of the particular costs of ingredient grains that are used to satisfy the nutritional requirements. The particular costs and requirements are the parameters of that specific realization of the model.

It is important to note that a problem whose structure remains relatively constant but whose parameters change regularly and is solved on a periodic basis is particularly well suited to a decision support methodology. This does not mean that a problem which undergoes structural change periodically is not suited to decision support, but the architecture of the decision support tool
is more problematic. In the former case a decision support methodology such as that discussed in Cohen (1986) is an appropriate solution. However, in the latter case a matrix generation language is the more appropriate path. With the model well specified, and the parameter values in hand, the actual computer implementation of the model is possible. Then appropriate software can be accessed so that the model can be solved for the decision variables. The final stage in the paradigm is the implementation of the solution as specified by the values of the decision variables.

Of course this paradigm is not the only overview possible. For example, it may be desirable to iterate between the stage at which the decision variables are solved and the model is formulated. This allows managers to evaluate the proposed solution before implementation and modify the model to capture overlooked details or secondary issues.

Regardless of the details of the paradigm we can group the steps into four major divisions: setup, generation, solution, and reporting. Model setup comprises those steps that cannot be mechanized, namely, problem identification, model abstraction and formulation, and data collection.

The model generation phase can certainly be mechanized and is most likely the section that benefits most from computerization. Computer generation includes building the structural portion of the model from the mathematical abstraction and adding the realization specific parameters.

In the solution phase, the model is passed to an optimizer, or model solver, for solution of the decision variables. Finally, in the reporting phase, the values of the decision variables, obtained from the optimizer, are reported. This step benefits greatly from computerization.

The ability to translate the mathematical representation of the model into a computer realization in a simple way can be of significant value to the analyst. This becomes particularly evident when one considers the diversity of applications in the area of linear programming.

A Simple Product Mix Example

Consider the simple product mix example alluded to above. A feed company mixes wheat, bran, corn, and rye to form a cereal feed. The company can purchase the grains from several competing sources. The wheat can be purchased from suppliers in either North Dakota or California, the corn can be purchased from three separate sources in Iowa, the rye can be purchased from two sources in Kentucky, and the bran can be purchased from Nebraska. Each supplier provides grain with differing nutritional values at differing costs. The cereal manufacturer wants to determine the least cost mix that meets the nutrition requirements of the finished cereal mixture. The specific data are given in the data set that is saved by the following DATA step.

DATA SSS;
  INPUT NAME $8. COST VIT1 VIT2 VIT3 FIBER
  REQUIRE $; CARDS;
  ND WHEAT 10.0 3 1 .3 7 VIT1
  CA WHEAT 10.5 3 1 .3 7 VIT2
  NE BRAN 9.75 2.9 0 .3 9 VIT3
  I CORN-1 5.50 1 0 .5 5 FIBER
  I CORN-2 5.60 1 0 .4 6 FIBER
  I CORN-3 5.55 1 0 .5 3 .
  KE RYE-A 8.60 0 .5 .1 6 .
  KE RYE-B 6.67 0 .5 1 7 .
  REQUIRED . 5 3 1 10 ;

The linear program that represents this problem is easily specified. Let the subscript i denote grain and the subscript j denote nutrient. Let \( x_i \) denote the decision variables, namely, the quantity of grain \( i \) that is to be used in the cereal mixture, \( c_i \) denote the cost of a unit of grain \( i \), \( a_{ij} \) denote the amount of nutrient \( j \) in a unit of grain \( i \), and \( r_j \) denote the amount of nutrient \( j \) required in the mixture.

Then the model is written as:

\[
\begin{align*}
\min & \quad \sum_{i=1}^{8} c_i x_i \\
\text{subject to:} & \quad \sum_{j=1}^{5} a_{ij} x_i \geq r_j \quad \text{for } j=1, \ldots, 4 \\
& \quad x_i \geq 0 \quad \text{for } i=1,2, \ldots, 8 .
\end{align*}
\]

The objective function is given in the first line. It expresses the objective in terms of the decision variables and says to minimize the total cost of the cereal mixture. The next four inequalities form the structural constraints that express the nutritional requirements that the cereal mixture must meet. The last eight inequalities form the nonnegativity constraints that capture the requirement that only those mixtures having nonnegative amounts of grain need be considered. When taken together, the objective and constraints say to find the minimum cost mixture that meets the nutritional requirements.

The aim of matrix generation languages is to translate the abstract model into a computer representation. The closer the computer representation to the mathematical model description the quicker the computer representation can be built and the fewer the data handling errors.

Matrix Generation System Structure

The MATGEN system of matrix generation macros is composed of five functional types of macros which are executed sequentially.

First, the system is initialized using the MATGEN statement. This resets all the global variables and primes the system for loading of tables. Second, the data are loaded into the system. The LOAD statement initializes the loading. Several statements are used to identify
the subscript dictionaries, the data tables, and the decision variables, or unknowns, to the system. Third, the model building is initialized using the model statement. In this section the mathematical model is specified using the using the table, subscript, and unknown names defined in the LOAD section. Fourth, the model is solved for the values of the decision variables by the SOLVE statement. Finally, the solution is reported using the REPORT statement.

The program that follows uses the simple product mix example given in the last section. It shows how one can build the model, solve for the decision variables, and report the solution, using the MATGEN system.

MATGEN; /* INITIALIZE GENERATOR */
LOAD; /* LOAD THE DATA TABLES */
SUBSCRPT J DEMANDS(OR=8); VARPRIORIZE NAMESET=J;
SUBSCRPT I DEMANDS(OR=8); VARPRIORIZE NAMESET=I;
TABLE C DEMANDS NAMESET=C COL=I;
TABLE A DEMANDS NAMESET=A COL=I;
TABLE B DEMANDS NAMESET=B COL=J;
TABLE D DEMANDS NAMESET=D COL=J;
UNKNOWN X I ;
MODEL ( /* BUILD THE MODEL */
MIN : UNKNOWN=X{I} COEF=A{I,J} RELATION=GE RHS=R{J} )
FORALL J ;
SOLVE; /* SOLVE THE MODEL */
REPORT; /* REPORT THE PRIMAL SOLUTION */

The reader should note the similarity between the matrix generation system statements and the abstract representation of the model. Furthermore, the statements are for the most part self-explanatory. Note however, that the nonnegativity constraints are missing. The system assumes that these always apply and they need not appear explicitly. Before discussing the MATGEN statements in more detail consider the report that is produced by the REPORT statement. It contains a line for each decision variable which gives the value of the decision variable, the reduced cost associated with the decision variable, and some of the input data. Although the report is rather simple and contains only information about the decision variables the REPORT macro demonstrates the programming details that can be used to write a more extensive report.

```
DESCRIP _PRICE_ _VALUE_ _BASE_ _PROFIT_ _SUB_ _BOUND_ _SUBBOUND_ 
X KG WHEAT 10.00 3.00 30.00 0.00 0 . . 
X EA WHEAT 10.50 0.00 0.00 0.50 0 . . 
X NE BRAN 9.75 0.00 0.00 6.45 0 . . 
X J CORN-1 5.50 0.50 1.10 0.50 1 . . 
X J CORN+2 5.60 0.00 0.00 1.20 0 . . 
X I CORN+3 5.55 0.00 0.00 0.45 0 . . 
X I CORN-3 8.60 0.00 0.00 4.20 0 . . 
X I CORN-2 8.67 0.00 0.00 4.22 0 . . 
```

Statement Details

This section discusses the statements in greater detail. Each of these statements is a macro in the MATGEN system.

LOAD Statement

The LOAD statement tells the system that the statements that follow will load tables of data into the system. Basically, loading is the process of identifying symbols with matrices. Later, in the model building section data from tables can be used by referring to the appropriate symbols. Three types of objects can be loaded, subscript, tables, and unknowns.

The LOAD statement has the syntax:

```
LOAD;
```

SUBSCRPT Statement The SUBSCRPT statement has the syntax:

```
SUBSCRPT name DS= d VAR = v NUMBER = r;
```

The name is a single character that is to be associated with the data loaded by the statement. The d is the name of a SAS data set. In the SUBSCRPT macro the macro variable &DS is appended to the DATA step SET statement so that any options that are valid SET options can be used. The v names a single variable that is in the d data set. If more than one variable is given in v, then only the first is used. The r gives the number of observations to read from the d data set. When observation i is read, the value of the v variable is associated with the ith value of the name subscript. In this way a type of dictionary is built that associates each observation in the identified data set with a subscript. The current implementation can have at most 99 values for a subscript.

TABLE Statement The TABLE statement is similar to the SUBSCRPT statement. It associates a symbol with a two-dimensional table of data. The syntax of the statement is:

```
TABLE name DS= d VAR = v ROW = r COL = c;
```

The name is a single character that is to be associated with the data loaded by the statement. The d is the name of a SAS data set. In the TABLE macro the macro variable &DS is appended to the DATA step SET statement so that any options that are valid SET options can be used. The v identifies a variable list of variables that are in the d data set. The r gives the number of observations to read in the v data set. The c gives the number of variables in the v list. When observation i is read the value of the jth variable in the v list is associated with the value of the entry in the rth row and jth column of the name table. In this way a data table or matrix is built from the data in the d data set.

UNKNOWN Statement The UNKNOWN statement simply defines the decision variables to the system. It tells the system how to interpret the subscripts. The syntax of the statement is:

```
UNKNOWN name subscripts;
```

1206
The name is a single character that is the name of the unknown. The subscripts is a list of up to three single character subscripts for the unknown. The subscripts must be defined to the system using the SUBSCRIPT statement either before or after the UNKNOWN statement.

It is important to note that when an unknown is defined the order in which the subscripts are given is important. When building the model with the MODEL statements the system does no consistency checks on the subscript usage. However, when reporting the solution the REPORT macro assumes the subscript order given in the UNKNOWN statement when interpreting the decision variables.

MODEL Statement

After the data have been defined to the system with the LOAD statements the model is built. The MODEL statement initiates model building and has the syntax;

MODEL NAME=model_name ;

The model_name is the name assigned to the model to be built, by default it is DEFMOD. There are several statements that are used to build a model the syntax for each is similar.

In most of the MODEL statements there is an UNKNOWN= parameter. An example of the use of this parameter is UNKNOWN=x(i) as is shown above. In general,

UNKNOWN=x{ exp : exp : exp }

where expression is an expression that evaluates to a valid subscript on x. Each subscript on x is separated by a colon. Similarly, many of the MODEL statements contain the COEF= parameter. This parameter takes expressions in terms of the subscripts and tables that have been load with the LOAD statements.

MIN and MAX Statements The MIN and MAX statements are used to give the objective function.

MIN subscripts SUBSET=e UNKNOWN=x COEF=c ;

The MIN and MAX statements function similarly, one creates an objective to be minimized and the other creates an objective to be maximized. The MIN statement is equivalent to

\[ \sum_{e} c \times x \]

The subscripts identify the subscripts over which to sum. These subscripts should have been loaded using the SUBSCRIPT statement so that the limits of the summation are the product of the limits of each of the subscripts listed in subscripts. Note that there should be no spaces between subscripts in subscripts. As is shown, the subset e is a logical expression that limits the summation, if e is true then the appropriate subscript is included in the summation otherwise it is excluded. The unknown x and the coefficient c are as described above.

SUM Statement The SUM statement is similar to the MIN statement. It has the syntax

SUM s{ exp : exp : exp ;}

The statement builds the following set of constraints.

\[ \sum_{e} c \times x \text{ for all } \{ s2 : e2 \} \]

LBOUND and UBOUND Statements The LBOUND and UBOUND statements define the bounds on decision variables.

LBOUND subscripts SUBSET=e UNKNOWN=x BOUND=b ;

This generates the following set of constraints.

\[ x \geq b \text{ for all } \{ \text{subscripts : e} \} \]

These constraints are not explicitly included in the model, but are implicitly included as the upper or lower bound type.

INTEGER and BINARY Statements The INTEGER statement identifies those decision variables that can only take integer values. The BINARY statement identifies the zero-one decision variables, it has the same syntax as the INTEGER statement with the statement name changed to BINARY.

INTEGER subscripts SUBSET=e UNKNOWN=x ;

Each variable x with subscript in the set

\[ \{ \text{subscripts : e} \} \]

is restricted to take only integer values.

LASTROW Statement Sometimes constraints cannot be easily expressed with a single summation. In these cases the LASTROW macro can be used to tell the system not to advance to the next constraint when it encounters a new SUM statement. An example of its use is shown in the Examples section.

SOLVE Statement

The SOLVE statement initiates model solution with the LP procedure.

SOLVE OPTIONS=PROC LP options MODEL=name ;

PROC LP options can be passed through the macro and the specific model to be solved can be identified. By default, the DEFMOD is solved. The LP procedure saves the primal solution in the data set named PRIMAL.
REPORT Statement The REPORT statement includes no parameters. It syntax is simply:

REPORT ;

The report includes a printed summary of the solution of the last model that was SOLVED. Each line of the report is associated with a decision variable in the model. The DESCRPT column gives the description of the decision variable by first giving its name then the value of its subscripts. The _VALUE_ gives the value of the decision variable at optimality, the _PRICE_ gives the input objective coefficient, the _COST_ gives the variables contribution to the objective value, _R_COST_ is the reduced cost, and _UBOUND_ and _LBOUND_ are the input bounds on the variable.

Programming Details

The MATSUB system is not a complete system. It is designed to give the functionality of a matrix generator but not be so involved that the logic in the code would be difficult to follow and take a long time to decipher. As a result the system provides broad outlines of a direction that can be quickly and easily extended.

The design focuses on providing an architecture that is computationally efficient and can be extended to become a total system. The efficiency is gained by building the entire model in a single DATA step. This not only leads to computational efficiencies but also provides the model builder with other features of the DATA step.

There are several limitations. Some are structural and result from the program architecture and others merely result from omissions in an effort to simplify the code. The most apparent of this latter type is the limited error checking. This area can be improved significantly without major modifications of the existing logic. For example, it is possible in the MODEL step to subscript unknowns or tables beyond the range declared for the table or unknown in the LOAD statements. The macro system does not identify this error. However, the error is identified during the execution of the DATA step associated with the MODEL statement. This type of error can be trapped and reported during macro execution.

There are several structural limitations that require discussion. As mentioned above a single unknown can have at most three subscripts. To understand this consider the methods used. The MODEL macro builds a linear program using the sparse data format to PROC IP. The unknowns are coded so that the first character is a number from one to nine denoting the unknown, the next two characters is a number 0 to 99 denoting the first subscript, the next two character is a number 0 to 99 denoting the second subscript, and the last two characters is a number 0 to 99 denoting the last subscript. In this way a unique column name is defined for each unknown. This coding scheme can be modified so that more subscripts can be included with a smaller range or fewer subscripts with a larger range. It is even possible for some subscripts to have varying ranges. These additions are not difficult, but would obfuscate the program and defeat the purpose of this paper.

Another structural limitation that can be circumvented is the limit on the size of a table. The current implementation requires that the total number of elements in all the tables fit into the program data vector. This means that if the variables have length eight then there can be at most approximately 4000 entries. However, if entries in tables do not need the full length precision than more entries can be obtained. It should be noted that this limit is for table entries and not for model variables. Since the sparse input to PROC LP is used, a model with more than 4000 variables can be built. As a result, multiple pieces of a large model can be built separately, without exceeding the 4000 table size limit, then put together.

Let us consider the macro system in more detail. Each time a SUBSCRPT, UNKNOWN, or TABLE statement is executed three global macro variables are updated. When the SUBSCRPT or TABLE statements are executed a data set is created with the name Dn, where n is the name of the subscript or table being loaded. For example, when the SUBSCRIPT n macro is invoked the data set Dn is created, the macro variable NSUBS is incremented, the macro variable MSUBS has the name of the subscript appended to it, and the macro variable DSUBS has appended to it the following:

```
SET Dn; ARRAY Dn (_K) 1-m;
```

where n is the name of the subscript and m is the number of elements. For the example given above %PUT &DSUBS &MSUBS &NSUBS; results in

```
SET DJ; ARRAY DJ (_K) J1-J3;
SET DI; ARRAY DI (_K) I1-I8; J1
```

In this example the data sets DJ and DI are created with variables J1 to J3 and I1 to I8, respectively. In the first observation these contain the values of the subscripts. These macro variables and the associated data set form a type of symbol table or data dictionary that is used by both the model building elements and the report writing elements of the system. The TABLE statement causes a similar updating of macro variables and building of output data sets. The two-dimensional table is placed into a one-dimensional array in row major order. These data are placed in a data set with one observation. The NTABLE macro variable is incremented, the NTABLE macro variable has the table name appended to it, and the DTABLE macro variable has appended to it the DATA step code needed to access the table entries. For the example given above, %PUT &DTABLE &MTABLE &NTABLE; yields:

```
ARRAY C8; SET DC;
ARRAY A32; SET DA;
ARRAY B4; SET DB;
CAM 3
```
The UNKNOWN statement causes a similar updating of macro variables but no data sets are created. %PUT &DUNKS &MUNKS &NUNKS; yields

ARRAY X (...) DI;
X 1

These macros will be used in the MODEL and REPORT statements. As mentioned, the MODEL statement is an alias for the DATA step. The DTABLE macro variable includes all the table arrays that have been defined to the system. Each of the macros that are invoked under the MODEL statement then has access to these data. The several MODEL statement macros call a common set of macros to actually do the work. Examination of the system of macros in the Appendix shows that most of these call the SIGMA macro. This macro encodes the unknown, parses the expressions associated with the RHS=, COEF= and SUBSET= parameters, and calls the LOOP macro to build the necessary DATA step DO loops.

The PARSE macro parses the expressions and converts the two-dimensional table references in the calling parameters into one-dimensional array references.

The SOLVE and REPORT macros have relatively straightforward logic. The REPORT macro uses the DSUBS, MUNKS, and DUNKS macros to decode the encoded unknown. This can be expanded to produce more elaborate reports by extending the symbol table logic that is given.

Example

In this section we present an example that is somewhat more involved than that given above. Here, we consider a model that is a generalized network. We model a multicommodity transshipment problem with arc gains. Suppose we have several commodities, denoted with the subscript k, that are to be shipped across a network. Let the subscripts i and j denote the nodes in the network and let

\[ X_{ijk} \text{ amount of } k \text{ to ship from } i \text{ to } j \]
\[ c_{ijk} \text{ cost of shipping one unit of } k \text{ from } i \text{ to } j \]
\[ d_{jk} \text{ demand for } k \text{ at } j \]
\[ s_{ik} \text{ supply of } k \text{ at } i \]
\[ r_{ij} \text{ rate of spoilage of } k \text{ on route } ij \]
\[ S(k) \text{ set of supply nodes of } k \]
\[ D(k) \text{ set of demand nodes for } k \]
\[ T(k) \text{ set of transshipment nodes for } k \]

The objective is to satisfy demand at D(k) for each k with the supply S(k) at each k at minimum cost. The formulation for this very general model is given follows.

\[ \min \sum_{i,j,k} c_{ijk} x_{ijk} \]
subject to
\[ \sum_{j} x_{ijk} = s_{ik} \text{ for all } i \text{ and } k \]
\[ \sum_{i} x_{ijk} = d_{jk} \text{ for all } j \text{ and } k \]
\[ x_{ijk} \geq 0 \text{ for all } i, j, \text{ and } k \]

To demonstrate how the MATGEN system is used to represent this model we consider a very small example with three commodities, apples, avocados, and pears. The network over which these commodities are shipped includes eight cities as nodes. These are given in the CITY data set shown below. The supply, demand, cost, and spoilage rates can also be found in the data set shown below.

**DATA CITY:**
**INPUT NAME $ @@; CARDS:**
SEATTLE SAN_FRAN SAN_DIGO
DENVER DALLAS NY WASH MIAMI

**DATA FRUIT:**
**INPUT NAME $ @@; CARDS:**
APPLE AVOCADO PEAR

**DATA SUPDEM:**
**INPUT CITY $ APPLES AVOCADOS PEARS**
APPLED AVOCADOD PEARD

**DATA COST:**
**INPUT CITY $ SEATTLE...**

**DATA SPOIL:**
**INPUT CITY $ SEATTLE SAN_FRAN SAN_DIGO**

...
Although the data given are not extensive, the MATGEN program that generates the appropriate linear program would be virtually identical even if there were 50 cities and thirty products. The only difference would be a change in the number of rows and columns.

MATGEN:

LOAD:

SUBSCRIPT 1 CITY VAR=NAME NUMBER=;
SUBSCRIPT 2 CITY VAR=NAME NUMBER=;
TABLE 3 NAME VARCHAR=APPLES..PEARS ROW=3 COL=3;
TABLE 4 CITY VARCHAR=NAMES ROW=8 COL=3;
TABLE 5 C VARCHAR=BATTLIE..MAY ROW=3 COL=3;
UNKNOWN X IJK;

MIN
SUM J
SUM I
DO K=1 TO 3;
SUM J SUBSET={1:J} = UNKNOWN {1:J} RHS=D{1:J};
DO I=1 TO 8;
SUM J SUBSET={1:J} = UNKNOWN {1:J} RHS=D{1:J};
END;
END;

SOLVE;

REPORT;

One reason for including this model is to show the construction of the last constraint. The LASTROW statement is needed to represent the constraint with the two summations. Normally, the SUM statement advances to the next constraint. The model also shows the use of the DATA step DO statements in conjunction with the MATGEN system. Because all model building is done in a single DATA step many of the programming features of the DATA step can be used under the MATGEN MODEL statement.

The relation RELATION= and COEF= parameters are missing on the first SUM statement. By default, the relation in LE and the coefficient is 1. If one of the relation=EQ were left off the SUM statement within the DO loop, then each of the constraints defined there would be defined as both LE and EQ type constraints. Because of the minimal error checking the macro system would not pick that up. However, PROC LP would identify those constraints as ambiguously defined.

The report generated by the REPORT statement is given below. It includes a line for each decision variable x_{ijk}. For each of these the values of the subscripts are given.

---

### Appendix

The set of MATGEN macros is reproduced here.

---

Conclusion

In this paper we have presented a system of SAS macros that can be used for matrix generation for linear programming problems. Although useful as is, the system is designed to be expanded and built upon. We have focused on the use of the system and briefly discussed the structure of the macros.
---

**THE LOAD PHASE CODE**

\[ \text{MACRO LOAD} / \text{STMT} : \]
\[ \text{LET PHASE=LOAD;} \]
\[ \text{LET NUM=NAME;} \]
\[ \text{DATA NAME} \ ( \text{KEEP } = \_\text{TYPE}, \_\text{COL}, \_\text{ROW}, \_\text{COEF}) ; \]
\[ \text{LENGTH} \ _\text{COL} $ \_\text{ROW} $ \_\text{COEF} $ \_\text{TYPE} $ \_\text{CSET} ; \]
\[ \text{RETURN} \ _\text{ROW} \ _\text{FLAG} \_\text{0} ; \]
\[ \text{ERROR} ; \]

\[ \text{MACRO UNKNOWN} (\text{name}, \text{SUBS}) / \text{STMT} : \]
\[ \text{IF} \ _\text{PHASE}=\text{LOAD} \ _\text{THEN} \ _\text{DO} ; \]
\[ \text{IF} \ _\text{PHASE}=\text{LOAD} \ _\text{THEN} \ _\text{DO} ; \]
\[ \text{ELSE} \ _\text{PUT} \ _\text{ERROR} : \text{UNKNOWN} \ _\text{IS} \ _\text{A} \ _\text{LOAD} \ _\text{PHASE} \ _\text{STATEMENT} . \]
\[ \text{END} ; \]

\[ \text{ELSE} \ _\text{PUT} \ _\text{ERROR} : \text{SUBSCRPT} \ _\text{IS} \ _\text{A} \ _\text{LOAD} \ _\text{PHASE} \ _\text{STATEMENT} . \]
\[ \text{END} ; \]

\[ \text{MACRO SUBSCRPT} (\text{name}, \text{DS}=, \text{VAR}=, \text{NUMBER}=) / \text{STMT} : \]
\[ \text{LET DSUBS}: = \text{EVAL} (\text{DS} \ _\text{SET} \ _\text{D}\ _\text{NUMBER}) ; \]
\[ \text{LET ENAME}=\text{EVAL} (\text{ENAME} \ _\text{NUMBER}) ; \]
\[ \text{LET SNAME}=\text{UPCASE} (\text{NAME}) ; \]
\[ \text{LET TEMP}=- \ _\text{UNQUOTE} (\text{TEMP}) ; \]
\[ \text{LET STATE}=\text{SUBSCRPT} ; \]
\[ \text{DATA DNAME} ; \]
\[ \text{SET ASS} ; \]
\[ \text{ARRAY NAME} (J\_1) \ _\text{ASNAME} \ _\text{NAME} ; \]
\[ \text{RETAIN NAME} = \ _\text{NAME} ; \]
\[ \text{KEEP NAME} = \ _\text{NAME} ; \]
\[ \text{ARRAY TEMP} (II) \ _\text{AVAR} II \ = \ _\text{NAME} \ _\text{NAME} ; \]
\[ \text{IF} \ _\text{NAME} \ _\text{NAME} \ _\text{NAME} \ _\text{NAME} \ _\text{NAME} \ _\text{NAME} ; \]
\[ \text{RETURN} \ _\text{TEMP} \ _\text{STATE} ; \]
\[ \text{END} ; \]

**THE MODEL PHASE CODE**

\[ \text{MACRO MODEL} (\text{NAME} , \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET NUM=NAME;} \]
\[ \text{DATA NAME} \ ( \text{KEEP } = \_\text{TYPE}, \_\text{COL}, \_\text{ROW}, \_\text{COEF}) ; \]
\[ \text{LENGTH} \ _\text{COL} $ \_\text{ROW} $ \_\text{COEF} $ \_\text{TYPE} $ \_\text{CSET} ; \]
\[ \text{RETURN} \ _\text{ROW} \ _\text{FLAG} \_\text{0} ; \]
\[ \text{ERROR} ; \]

\[ \text{MACRO UBOUND} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=BOUND} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE BINARY PHASE CODE**

\[ \text{MACRO INTEGER} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=INTEGER} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE SUM PHASE CODE**

\[ \text{MACRO SUM} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=SUM} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE MAX PHASE CODE**

\[ \text{MACRO MAX} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=MAX} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE MIN PHASE CODE**

\[ \text{MACRO MIN} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=MIN} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE LASTROW PHASE CODE**

\[ \text{MACRO LASTROW} / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{IF} \ _\text{FLAG} \ _\text{THEN} \ _\text{RETURN} \ _\text{TEMP} ; \]
\[ \text{ERROR} ; \]

**THE UBOUND PHASE CODE**

\[ \text{MACRO UBOUND} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=UBOUND} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE INTEGER PHASE CODE**

\[ \text{MACRO INTEGER} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=INTEGER} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE SUM PHASE CODE**

\[ \text{MACRO SUM} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=SUM} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE MAX PHASE CODE**

\[ \text{MACRO MAX} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=MAX} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE MIN PHASE CODE**

\[ \text{MACRO MIN} (\text{SUBSCRPT}, \text{SUBSET}=, \text{UNKNOWN}=, \text{BOUND}=) / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{LET STATE=MIN} ; \]
\[ \text{RETURN} \ _\text{STATE} ; \]

**THE LASTROW PHASE CODE**

\[ \text{MACRO LASTROW} / \text{STMT} : \]
\[ \text{LET PHASE=MODEL;} \]
\[ \text{IF} \ _\text{FLAG} \ _\text{THEN} \ _\text{RETURN} \ _\text{TEMP} ; \]
\[ \text{ERROR} ; \]
/* MACRO FOR(SUBSCRIPT, SUBSET=I=1, ACTION=1); */
SPARSE(ARG=SUBSCRIPT, RET=SUBSET);
LOOP((I SUBSCRIPT, RET =Flag)); IF SUBSET THEN DO; ACTION; END;
END;

/* MACRO SIGMA(SUBSCRIPT, SUBSET=I=1, UNKNOWN=1, COEF=1, RELATION=LE, RHS=0); */
IF SUBSCRIPT = THEN ERROR(1, SUBSCRIPT);
ELSE IF UNKNOWN = THEN ERROR(1, UNKNOWN);
LET UNKNOWN = UPPCASE(UNKNOWN);
/* ENCODE THE UNKNOWN */
LET I = INDEX(UNKNOWN, SUBSTR); LET TEMP = (1 + I) / 2;
LET TRANS = (SUBSTR(I, 1, I)); LET TRANS = LEFT(PUT(SUBSTR, 5, ), )
LET TRANS = LEFT(5, RHS ); LET TEMP = (SUBSTR(I, 2, )); LET I = EVAL(1 + I / 2);
END;

/* MACRO PARSE(ARG=ARG, RET=RET); /"" LOOKS FOR •.• A EXP EXP ••• */
LET ARG = UPPCASE(ARG);
LET &RET = ;
DO WHILE (LENGTH(ARG));
LET &RET = &RET . SUBSTR(ARG, 1, I);
LET I = EVAL(LENGTH(ARG) - I);
LET TEMP = SUBSTR(&&&.RET , I, 1);
LET &RET = T(DECVAR); LET I = INDEX(ARG, );
END;
LET &RET = UPPQUOTE(&&&.RET &ARG);
END;

/* MACRO LOOP(I=1, DO=); */
LET CONTRL = ; LET CONTRL = ;
References


SAS and SAS/OR are registered trademarks of SAS Institute Inc., Cary, N.C.