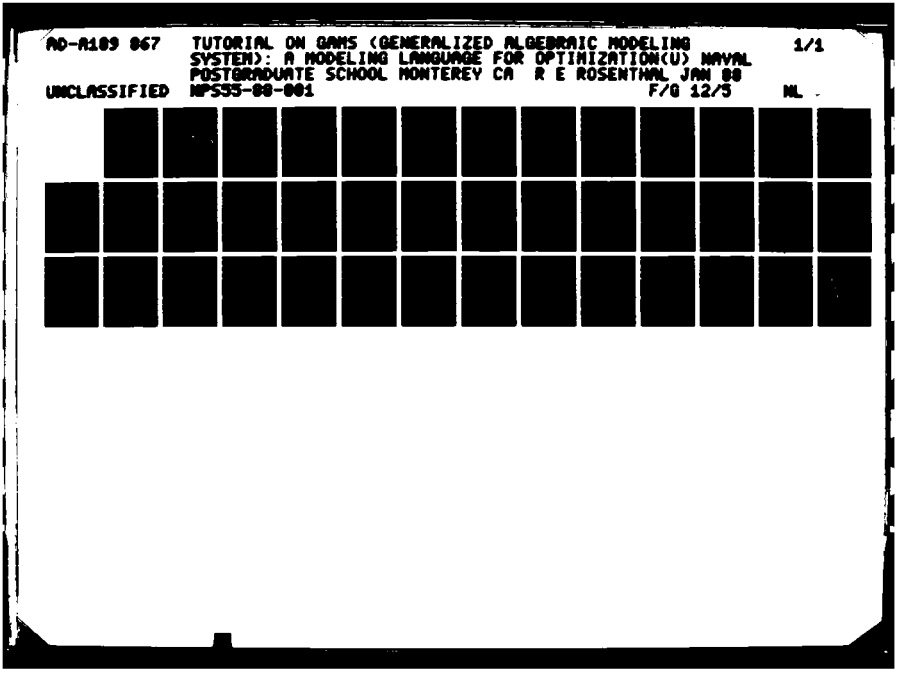
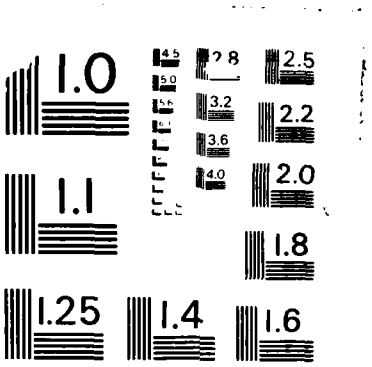

AD-A189 867
TUTORIAL ON GAMS (GENERALIZED ALGEBRAIC MODELING SYSTEM): A MODELING LANGUAGE FOR OPTIMIZATION(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R E ROSENTHAL JAN 88
1/1
UNCLASSIFIED
NPS55-88-001
F/G 12/5
NL



NPS55-88-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

TUTORIAL ON GAMS: A MODELING LANGUAGE FOR OPTIMIZATION

RICHARD E. ROSENTHAL

JANUARY 1988

Prepared for:
Naval Postgraduate School
Monterey, CA 93943-5000

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

Rear Admiral R. C. Austin
Superintendent

K. T. Marshall
Acting Provost

Reproduction of all or part of this report is authorized.

This report was prepared by:

Richard E. Rosenthal

RICHARD E. ROSENTHAL
Associate Professor of
Operations Research

Reviewed by:

P. Purdue

PETER PURDUE
Professor and Chairman
Department of Operations Research

Released by:

JAMES T. FREMGEN
Acting Dean of Information and
Policy Sciences

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS55-88-001	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 55
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
8c ADDRESS (City, State, and ZIP Code)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
TUTORIAL ON GAMS: A MODELING LANGUAGE FOR OPTIMIZATION

12 PERSONAL AUTHOR(S)
Rosenthal, Richard E.

13a TYPE OF REPORT Technical Report	13b TIME COVERED FROM ___ TO ___	14 DATE OF REPORT (Year Month Day) 1988 Jan	15 PAGE COUNT 39

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Optimization, Modeling Languages, Mathematical Programming, GAMS

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The Generalized Algebraic Modeling System (GAMS), developed by Alexander Meeraus, Anthony Brooke et al., is a language for expressing mathematical models in a form that is both easily understandable to a modeler and processible by a computer. For practitioners and researchers, it can greatly reduce the time required for formulating, documenting, solving and analyzing large, complex linear, nonlinear and integer programs. For teachers and students, it is a valuable tool for learning advanced modeling. This paper is a tutorial for the beginning GAMS user.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Richard E. Rosenthal	22b TELEPHONE (Include Area Code) (408)646-2795 / 22c OFFICE SYMBOL Code 55Rl

DD FORM 1473, 84 MAR ... PREVIOUS EDITIONS ARE OBSOLETE ... SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

A GAMS TUTORIAL

Introduction

This book[1] begins with a simple example of the use of GAMS for formulating, solving and analyzing an optimization problem. The example is an instance of the transportation problem of linear programming, which has historically served as a "laboratory animal" in the development of optimization technology. It is a good choice for illustrating the power of algebraic modeling languages like GAMS, because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices:
- i = plants
- j = markets

Given Data:
- a_i = supply of commodity at plant i (in cases)
- b_j = demand for commodity at market j (cases)
- c_{ij} = cost per unit shipment between plant i and market j ($/case)

Decision Variables:
- x_{ij} = amount of commodity to ship from plant i to market j (cases), where $x_{ij} \geq 0$, for all i,j

---

[1] This paper will appear as a chapter in the forthcoming book, **GAMS: A User's Guide**, by Anthony Brooke, David Kendrick and Alexander Meeraus, to be published in 1988 by The Scientific Press. The tutorial refers to other chapters of the forthcoming book as a source of additional information, but is otherwise self-contained.)

Constraints:

Observe supply limit at plant i:

$$\Sigma_j x_{ij} \leq a_i, \quad \text{for all } i \qquad \text{(cases)}$$

Satisfy demand at market j:

$$\Sigma_i x_{ij} \geq b_j, \quad \text{for all } j \qquad \text{(cases)}$$

Objective Function:

$$\text{minimize } \Sigma_i \Sigma_j c_{ij} x_{ij} \qquad (\$K)$$

We remark here that this simple example reveals some modeling practices which we regard as good habits in general and which are consistent with GAMS's design. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol $K here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms "exogenous variable" and "endogenous variable" for "given data" and "decision variable," respectively. In GAMS, the terminology adopted is as follows: indices are called SETS, given data are called PARAMETERS, decision variables are called VARIABLES, and constraints and the objective function are called EQUATIONS.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS form can be read and processed by a computer.

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the given data as follows [Ref. Dantzig].

	Markets			
	New York	Chicago	Topeka	
Plants	Shipping Distances			Supplies
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
Demands	325	300	275	

Shipping distances are in thousands of miles and shipping costs are assumed to be $90.00 per case per thousand miles.

The GAMS representation of this problem is as follows:

```
SETS
     I   canning plants   / SEATTLE, SAN-DIEGO /
     J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;

PARAMETERS

     A(I)  capacity of plant i in cases
         /  SEATTLE     350
            SAN-DIEGO   600  /

     B(J)  demand at market j in cases
         /  NEW-YORK    325
            CHICAGO     300
            TOPEKA      275  / ;

TABLE D(I,J)  distance in thousands of miles
                  NEW-YORK       CHICAGO       TOPEKA
    SEATTLE          2.5            1.7           1.8
    SAN-DIEGO        2.5            1.8           1.4  ;

SCALAR F  freight in dollars per case per thousand miles  /90/ ;

PARAMETER C(I,J)  transport cost in thousands of dollars per case ;

          C(I,J) = F * D(I,J) / 1000 ;

VARIABLES
     X(I,J)  shipment quantities in cases
     Z       total transportation costs in thousands of dollars ;

POSITIVE VARIABLE X ;

EQUATIONS
     COST        define objective function
     SUPPLY(I)   observe supply limit at plant i
     DEMAND(J)   satisfy demand at market j ;

COST ..        Z  =E=  SUM((I,J), C(I,J)*X(I,J)) ;

SUPPLY(I) ..   SUM(J, X(I,J))  =L=  A(I) ;

DEMAND(J) ..   SUM(I, X(I,J))  =G=  B(J) ;

MODEL TRANSPORT /ALL/ ;

SOLVE TRANSPORT USING LP MINIMIZING Z ;
```

```
DISPLAY X.L, X.M ;
```

If you submit a file containing the statements above as input to the GAMS program, the transportation model will be formulated and solved. Appendix A gives detailed instructions on how to invoke GAMS on several kinds of computers, but the simplest ("no frills") way to call GAMS is to enter the word GAMS followed by the input file's name[2]. If all goes well you will receive copious output from GAMS, at the bottom of which will be the optimal shipments displayed as follows.

	NEW-YORK	CHICAGO	TOPEKA
SEATTLE	[illegible]	300.000	
SAN-DIEGO	[illegible]		275.000

You will also receive the marginal costs (simplex multipliers) below.

	CHICAGO	TOPEKA
SEATTLE		0.[illegible]6
SAN-DIEGO	[illegible]	

These numbers indicate, for example, that it is optimal to send nothing from [illegible] to Topeka, but if you insist on sending one case it will add [illegible] (for $36.00) to the optimal cost. (Can you prove this figure without more data than the optimal shipments and the given data?[3])

1. Structure of a GAMS Input File

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are:

---

[2] When running GAMS you usually omit the file's type or extension, but you may be required to use certain choices. For example, on DOS-based computers, the input file's extension must be ".GMS," and on the VM operating system, the input file's type must be "GAMS."

[3] If you [illegible] to Topeka then, to maintain supply/demand [illegible] one less case from San Diego to Topeka, one more case from Seattle to New York, and one less case from Seattle to Chicago. The net increase in shipping distance is +1800 -1400 [illegible], which costs $36 at the given shipping rate.

Inputs

- SETS
  - Declaration
  - Assignment of members
- Data (PARAMETERS, TABLES, SCALARS)
  - Declaration
  - Assignment of values
- VARIABLES
  - Declaration
  - Assignment of type
  - (Optional) assignment of bounds and/or initial values
- EQUATIONS
  - Declaration
  - Definition
- MODEL and SOLVE statements
- (Optional) DISPLAY statements

Outputs

- Echo Print
- Reference Maps
- Equation Listings
- Status Reports
- Results

There are optional input components such as edit checks for bad data, and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

1. A GAMS model is a collection of statements in the GAMS language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.

2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples of the tutorial, but precise rules of the road are given in Chapter ##.

3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The rules for relaxed punctuation are given in Chapter ##.

4. The GAMS compiler does not distinguish between upper and lower case letters, so you are free to use either. The style adopted and recommended here is to always use upper case for any word or symbol which is part of the GAMS language or is an entity declared to exist in a particular GAMS model. We recommend reserving lower case for words that appear in the GAMS input for documentation only.

5. Documentation is crucial to the usefulness of mathematical models. It is all the more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more importantly, documentary text can be inserted within specific GAMS statements. All the lower case words in the transportation model are examples of the second form of documentation.

6. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration, and an assignment or definition. "Declaration" means declaring the existence and giving a name to something. "Assignment" or "definition" means giving specific value or form to something. In the case of EQUATIONS, you must make the declaration and definition in separate GAMS statements. However, for all other GAMS entities you have the option of making declarations and assignments in the same statement or separately.

7. The names given to the entities of the model must start with a letter and can be followed by up to nine more letters or digits.

2. Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The transportation example above contains just one SET statement:

```
SETS
     I   canning plants    / SEATTLE, SAN-DIEGO /
     J   markets           / NEW-YORK, CHICAGO, TOPEKA / ;
```

The effects of this statement are probably self-evident. We declared two sets and gave them the names I and J. We also assigned members to the sets as follows:

I = (Seattle, San Diego)

J = (New York, Chicago, Topeka).

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes rather than curly braces to delineate the set, simply because not all computer keyboards have keys for curly braces. Note also that multi-word names like "New York" are not allowed, so hyphens are inserted.

The lower case words in the SETS statement above are called "text." Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and "parrots" it back to you at various times for your convenience.[4]

It was not necessary to combine the creation of sets I and J in one statement. We could have put them into separate statements as follows:

```
SET   I   canning plants   / SEATTLE, SAN-DIEGO / ;

SET   J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;
```

The placement of blank spaces and lines (as well as the choice of upper or lower case) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular SET or the plural SETS is also up to you. Using SET in a statement that makes a single declaration and SETS in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use sometimes when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid SET statements in GAMS.

```
SET   T   time periods   / 1991 * 2000 / ;

SET   M   machines       / MACH1 * MACH24 / ;
```

Here the effect is to assign

T = (1991, 1992, 1993, ..., 2000)

---

[4] The text must fit on one line and cannot exceed 80 characters in length. It should not start with one of GAMS's reserved words or contain any of the following special characters: equal sign, comma, semicolon or slash (= , ; /).

M = (MACH1, MACH2, ..., MACH24) .

Note that set elements are stored as character strings, so the elements of T are not numbers.

Another convenient feature is the ALIAS statement which is used to give another name to a previously declared set. In the following example,

ALIAS (T,TP) ;

the name TP is like a T-prime in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets I, J, T and M in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in Chapter ##. Another valuable advanced feature is multi-dimensional sets which are discussed in Chapter ##.

3. Data

The GAMS model of the transportation problem demonstrates all of the three fundamentally different formats that are allowable for entering data in GAMS. The three formats are:

1. lists,
2. tables, and
3. direct assignments.

3.1 Data Entry by Lists

The first format is illustrated by the first PARAMETERS statement of the example, which is repeated below.

```
PARAMETERS

     A(I)  capacity of plant i in cases
        /  SEATTLE     350
           SAN-DIEGO   600  /
```

```
B(J)  demand at market j in cases
    /  NEW-YORK     325
       CHICAGO      300
       TOPEKA       275  / ;
```

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names A and B, and declares their "domains" to be I and J, respectively. (A domain is the set (or tuple of sets) over which a GAMS parameter, variable or equation is defined.) The statement also gives documentary text for each parameter and assigns values of A(I) and B(J) for each element of I and J. It would have been perfectly acceptable to break this one statement into two, if you prefer, as follows.

```
PARAMETER  A(I)  capacity of plant i in cases
    /  SEATTLE     350
       SAN-DIEGO   600  / ;

PARAMETER  B(J)  demand at market j in cases
    /  NEW-YORK    325
       CHICAGO     300
       TOPEKA      275  / ;
```

Here are some points to remember when using the list format.

1. The list of domain elements and their respective parameter values can be laid out typographically in almost any manner you like. The only rules are that the entire list must be enclosed in slashes, and that the element-value pairs must be separated by commas or entered on separate lines.

2. There is no semicolon separating the element-value list from the name, domain, and text which precede it. That's because the same statement is being used for declaration and assignment when you use the list format. (An element-value list all by itself, is not interpretable by GAMS and will result in an error message.)

3. The GAMS compiler has an unusual feature called "domain checking," which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell Seattle correctly in the statement declaring SET I but misspell it as Seatle in a subsequent element-value list, the GAMS compiler would give you an error message that the element Seatle does not belong to the set I.

4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order.

5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a SCALAR statement containing a "degenerate" list of only one value, as in the following statement from the transportation model.

```
SCALAR F  freight in dollars per case per thousand miles  /90/ ;
```

If a parameter's domain has two or more dimensions, then it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few nonzeros) and supersparse (having few distinct nonzeros). This is discussed in Chapter ##.

3.2 Data Entry by Tables

It has been noticed for a long time among optimization practitioners that much of the input data for a large model is derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided in the transportation model:

```
TABLE D(I,J)  distances in thousands of miles
                 NEW-YORK       CHICAGO      TOPEKA
SEATTLE             2.5            1.7          1.8
SAN-DIEGO           2.5            1.8          1.4  ;
```

The effect of this statement is to declare the parameter D and to specify its domain as the set of ordered pairs in the Cartesian product of I and J. The values of D are also given in this statement, under the appropriate heading. If there are blank entries in a table they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in Chapter ##.

3.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method.

```
PARAMETER C(I,J)  transportation cost in dollars per case ;
          C(I,J) = F * D(I,J) ;
```

It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you a terse but helpful error message.)

The effects of the first statement above are to declare the parameter C, to specify the domain (I,J), and to provide some documentary text. The second statement assigns to C(I,J) the product of the values of the parameters F and D(I,J). Quite naturally, this is legal in GAMS only if you have already assigned values to F and D(I,J) in previous statements.

The direct assignment above applies to all (I,J) pairs in the domain of C. If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

```
C("SEATTLE","NEW-YORK") = 0.40 ;
```

is a valid GAMS assignment statement.

The same parameter can be assigned value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand-side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as FORTRAN, for example, you will have no trouble becoming immediately comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies not shared by FORTRAN. For example, we were able to assign C(I,J) values for all (I,J) pairs without constructing "do loops.")

The GAMS standard operations and supplied functions are given in Table ##. Here are some examples of valid assignments. In all cases, assume that in previous statements the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values.

```
CSQUARED = SQR(C) ;

E = M * CSQUARED ;

W = L / LAMDA ;

EOQ(I)  =  SQRT( 2 * DEMAND(I) * ORDCOST(I) / HOLDCOST(I) ) ;
```

```
T(I) = MIN( P(I), Q(I)/R(I), LOG(S(I)) ) ;

EUCLIDEAN(I,J) = SQRT(SQR(X1(I) - X1(J)) + SQR(X2(I) - X2(J))) ;

PRESENT(J) = FUTURE(J) * EXP( - INTEREST * TIME(J) ) ;
```

The summation and product operator to be introduced later can also be used in direct assignments.

4. Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a VARIABLES statement. Each variable is given a name, a domain if appropriate, and, optionally, text. The transportation model contains the following example of a VARIABLES statement.

```
VARIABLES
    X(I,J)  shipment quantities in cases
    Z       total transportation costs in thousands of dollars ;
```

This statement results in the declaration of a shipment variable for each i,j pair. (You will see in Chapter ## how GAMS can handle the typical real-world situation in which only a subset of the i,j pairs are allowable for shipment.)

The Z variable is declared without a domain, because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. The permissible types are give below.

Name of Variable Type	Allowed Range of Variable
FREE (default type)	$-\infty$ to $+\infty$
POSITIVE	0 to $+\infty$
NEGATIVE	$-\infty$ to 0
BINARY	0 or 1
INTEGER	0,1, ..., 100

The variable which serves as the quantity to be optimized MUST be a scalar and MUST be of the FREE type. In our transportation example, Z is kept free by default, but X(I,J) is constrained to nonnegativity by the following statement.

```
POSITIVE VARIABLE X;
```

Note that the domain of X should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

The method of assignment of lower bounds, upper bounds and initial values to variables is given in Section 9.1.

5. Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities which comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

5.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, EQUATIONS in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation example contains the following equation declaration:

```
EQUATIONS
     COST         define objective function
     SUPPLY(I)    observe supply limit at plant i
     DEMAND(J)    satisfy demand at market j ;
```

Keep in mind that the word "EQUATION" has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and with a single name it can refer to one or several of these relationships. For example, COST has no domain so it is a single equation, but SUPPLY refers to a set of inequalities defined over the domain I.

5.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe GAMS's notation for summations. Remember that GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would it be convenient for the user) to employ the standard mathematical notation for summations.

GAMS's summation notation can be used for simple and complex expressions. Its format is based on the idea of always thinking of a summation as an operator with two arguments:

```
SUM( index of summation, summand )
```

The two arguments are separated by a comma, and if the first argument requires a comma then it should be in parentheses. The second argument can be any mathematical expression including another summation.

As a simple example, the transportation problem contains the expression

```
SUM( J, X(I,J) )
```

which is equivalent to $\Sigma_j x_{ij}$.

A slightly more complex summation is used in the following example:

```
SUM( (I,J), C(I,J)*X(I,J) )
```

which is equivalent to $\Sigma_i \Sigma_j c_{ij}x_{ij}$.

The last expression could also have been written as a nested summation as follows:

```
SUM( I, SUM( J, C(I,J)*X(I,J) ) )
```

In Chapter ##, we describe how to use the "dollar" operator to impose restrictions on the summation operator so that only the elements of I and J which satisfy specified conditions are included in the summation.

Products are defined in GAMS using the exact same format as summations, replacing "SUM" by "PROD". For example,

```
PROD( J, X(I,J) )
```

is equivalent to $\pi_j x_{ij}$.

Summation and product operators may be used in direct assignment statements for parameters. For example,

```
SCALAR  TOTSUPPLY  total supply over all plants ;

TOTSUPPLY  =  SUM( I, B(I) ) ;
```

5.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of the amount of variety that is possible. The components of an equation definition are, in order:

1. the name of the equation being defined
2. the domain
3. (optional) domain restriction condition
4. the symbol " .. "
5. left-hand-side expression
6. relational operator: =L=, =E=, or =G=
7. right-hand-side expression.

The transportation example contains three of these statements.

```
COST ..        Z  =E=  SUM((I,J), C(I,J)*X(I,J)) ;

SUPPLY(I) ..   SUM(J, X(I,J))  =L=  A(I) ;

DEMAND(J) ..   SUM(I, X(I,J))  =G=  B(J) ;
```

Here are some points to remember.

1. The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the DEMAND definition will result in the creation of one constraint for each element of the domain J, as shown in the following excerpt from the GAMS output.

```
DEMAND(NEW-YORK)..  X(SEATTLE,NEW-YORK) + X(SAN-DIEGO,NEW-YORK) =G= 325 ;

DEMAND(CHICAGO)..  X(SEATTLE,CHICAGO) + X(SAN-DIEGO,CHICAGO) =G= 300 ;

DEMAND(TOPEKA)..  X(SEATTLE,TOPEKA) + X(SAN-DIEGO,TOPEKA) =G= 275 ;
```

The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)

2. In many real-world problems, some of the members of an EQUATION's domain need to be omitted or altered from the pattern of the others due to an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as

the "dollar" or "such-that" operator, which is not illustrated here but is discussed in Chapter ##. The domain restriction feature can be absolutely essential for keeping the size of a real-world model down in the range of solvability.

3. The relational operators have the following meanings:

=L=	less than or equal to
=G=	greater than or equal to
=E=	equal to

It is important to understand the difference between the symbols "=" and "=E=". The "=" symbol is used only in direct assignments and the "=E=" symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definitior also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.

4. Variables can appear on the left- or right-hand-side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.

5. An equation definition can appear anywhere in the GAMS input provided the equation and all variables and parameters it refers to are previously declared. (Note, it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they are declared.

6. Objective Function

This is just a reminder that GAMS has no explicit entity called the "objective function." To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain), and which appears in an equation definition that equates it to your objective function.

7. MODEL and SOLVE Statements

The word "MODEL" in GAMS has a very precise meaning. It is simply a collection of EQUATIONS. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the

keyword MODEL followed by the model's name, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter /ALL/ in place of the explicit list. In our example, there is one MODEL statement:

```
MODEL  TRANSPORT  /ALL/ ;
```

This statement may seem superfluous to you, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut /ALL/, the statement would be written as

```
MODEL  TRANSPORT  / COST, SUPPLY, DEMAND / ;
```

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprise a specific model (or submodel) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a SOLVE statement, which in our example is written as

```
SOLVE TRANSPORT USING LP MINIMIZING Z ;
```

The format of the SOLVE statement is as follows:

1. the keyword "SOLVE"
2. the name of the model to be solved
3. the keyword "USING"
4. an available solution procedure, such as
   "LP" for linear programming
   "NLP" for nonlinear programming
   "MIP" for mixed integer programming, or
   "RMIP" for relaxed mixed integer programming
5. the keyword "MINIMIZING" or "MAXIMIZING"
6. the name of the variable to be optimized.

8. DISPLAY Statements

The SOLVE statement will cause several things to happen when it is executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the solver's output will be printed to a file. To get the optimal values of the primal and/or dual variables we can look at the solver's output, or, if we desire, we can request a display of these results from GAMS. Our example contains the following statement

```
DISPLAY X.L, X.M ;
```

which calls for a printout of the final levels, X.L, and marginals (or reduced costs), X.M, of the shipment variables X(I,J). GAMS will automatically format this printout in two-dimensional tables with appropriate headings.

9. The " .LO, .L, .UP, .M " Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. There are four fields in each record:

```
.LO  =  lower bound
.L   =  level or primal value
.UP  =  upper bound
.M   =  marginal or dual value
```

The format for referencing these quantities is: the variable or equation's name, followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read- and write-access to the database. This may not seem remarkable to you now but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.

9.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (FREE, POSITIVE, NEGATIVE, BINARY or INTEGER), but these bounds can be overwritten by the GAMS user. Some examples follow.

```
X.UP(I,J) = CAPACITY(I,J) ;
X.LO(I,J) = 10.0 ;
X.UP("SEATTLE","NEW-YORK") = 1.2 * CAPACITY("SEATTLE","NEW-YORK") ;
```

It is assumed in the first and third examples that CAPACITY(I,J) is a parameter that was previously declared and assigned value. These statements must appear after the variable declaration and before the SOLVE statement. All the mathematical expressions available for direct assignments are usable on the right-hand-side.

In nonlinear programming, it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bounds. It is also very helpful to specify an

initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are QUANTITY(I), and it is known that the optimal solution to the unconstrained version of the problem is a parameter called EOQ(I). As a guess for the optimum of the constrained problem we enter

```
QUANTITY.L(I) = 0.5 * EOQ(I) ;
```

(The default initial level is the lower bound (.LO) if finite, otherwise it is zero.)

It is important to understand that the .LO and .UP fields are entirely under the control of the GAMS user. The .L and .M fields, in contrast, can be initialized by the user but are then controlled by the solver.

9.2 Transformation and Display of Optimal Values

(This section can be skipped on first reading if desired.)

After the optimizer is called via the SOLVE statement, the values it computes for the primal and dual variables are placed in the database in the .L and .M fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the SOLVE statement, we would enter

```
PARAMETER  PCTX(I,J)  per cent of market j's demand filled by plant i

PCTX(I,J)  =  100.0 * X.L(I,J) / B(J) ;

DISPLAY PCTX ;
```

Appending these commands to the original transportation problem input results in the following output:

```
                PER CENT OF MARKET J'S DEMAND FILLED BY PLANT I

                  NEW-YORK      CHICAGO       TOPEKA

SEATTLE             15.385      100.000
SAN-DIEGO           84.615                    100.000
```

For an example involving marginals, we briefly consider the "ratio constraints" that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let

Y(I,J) be the variable for the number of tons of raw material i put into finished product j. Let Q(J) be the variable for the number of tons of product j produced. Suppose the "ratio constraint" is that no product can consist of more than 25% of one ingredient, that is,

```
Y(I,J) / Q(J)  =L=  .25
```

for all i,j. To keep the model linear, the constraint is written as:

```
RATIO(I,J)..  Y(I,J) - .25 * Q(J)  =L=  0.0 ;
```

rather than explicitly as a ratio.

The problem here is that RATIO.M(I,J), the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

```
Y(I,J) - .25 * Q(J)  =L=  1.0 ;
```

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ratio constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

```
Y(I,J) / Q(J)  =L=  .26 ;
```

We can, in fact, obtain the desired marginals by entering the following transformation on the undesired marginals:

```
PARAMETER  AMR(I,J)  appropriate marginal for ratio constraint ;

AMR(I,J)  =  RATIO.M(I,J) * 0.01 * Q.L(J) ;

DISPLAY AMR ;
```

Notice that the assignment statement for AMR accesses both .M and .L records from the database. The idea behind the transformation is to notice that

```
Y(I,J) / Q(J) =L= .26 ;
```

is equivalent to

```
Y(I,J) - .25 * Q(J)  =L=  0.01 * Q(J) ;
```

10. GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion see Chapter ##. This tutorial discusses output partially as follows:

Outputs

Echo Print
Reference Maps
Error Messages

or

Echo Print
Reference Maps
Equation Listings
Model Statistics
Status Reports
Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the unfair impression that flawless use of advanced software is supposed to be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users have the capacity to make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

10.1 Echo Print

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand-side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

```
 3
 4   SETS
 5        I   canning plants   / SEATTLE, SAN-DIEGO /
 6        J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;
 7
 8   PARAMETERS
 9
10        A(I)  capacity of plant i in cases
11              /  SEATTLE     350
12                 SAN-DIEGO   600  /
```

```
 13
 14        B(J)  demand at market j in cases
 15            /  NEW-YORK     325
 16               CHICAGO      300
 17               TOPEKA       275  / ;
 18
 19   TABLE D(I,J)  distance in thousands of miles
 20                       NEW-YORK       CHICAGO       TOPEKA
 21       SEATTLE            2.5            1.7           1.8
 22       SAN-DIEGO          2.5            1.8           1.4  ;
 23
 24   SCALAR F  freight in dollars per case per thousand miles  /90/ ;
 25
 26   PARAMETER C(I,J)  transport cost in thousands of dollars per case ;
 27
 28             C(I,J) = F * D(I,J) / 1000 ;
 29
 30   VARIABLES
 31        X(I,J)  shipment quantities in cases
 32        Z       total transportation costs in thousands of dollars ;
 33
 34   POSITIVE VARIABLE X ;
 35
 36   EQUATIONS
 37        COST         define objective function
 38        SUPPLY(I)    observe supply limit at plant i
 39        DEMAND(J)    satisfy demand at market j ;
 40
 41   COST ..         Z  =E=  SUM((I,J), C(I,J)*X(I,J)) ;
 42
 43   SUPPLY(I) ..    SUM(J, X(I,J))  =L=  A(I) ;
 44
 45   DEMAND(J) ..    SUM(I, X(I,J))  =G=  B(J) ;
 46
 47   MODEL TRANSPORT /ALL/ ;
 48
 49   SOLVE TRANSPORT USING LP MINIMIZING Z ;
 50
 51   DISPLAY X.L, X.M ;
```

The reason why this echo print starts with line number 3 rather than line number 1 is because the input file contains two "dollar-print-control" statements. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1. The two used in our example are as follows.

```
$TITLE  A TRANSPORTATION MODEL
$OFFUPPER
```

The $TITLE statement causes the subsequent text to be printed at the top of each page of output. The $OFFUPPER statement is needed in order for the echo to contain mixed upper and lower case. Other available instructions are given in Chapter ##.

10.2 Reference Maps

The second section of output, whether the run terminates with errors or not, is a pair of "reference maps" which contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a "cross-reference map" such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows.

SYMBOL	TYPE	REFERENCES					
A	PARAM	DECLARED	10	DEFINED	11	REF	4
B	PARAM	DECLARED	14	DEFINED	15	REF	4
C	PARAM	DECLARED	26	ASSIGNED	28	REF	4
COST	EQU	DECLARED	37	DEFINED	41	IMPL-ASN	4
		REF	47				
D	PARAM	DECLARED	19	DEFINED	19	REF	2
DEMAND	EQU	DECLARED	39	DEFINED	45	IMPL-ASN	4
		REF	47				
F	PARAM	DECLARED	24	DEFINED	24	REF	2
I	SET	DECLARED	5	DEFINED	5	REF	1
		19	26	28	31	38	2*4
		2*43	45	CONTROL	28	41	4
		45					
J	SET	DECLARED	6	DEFINED	6	REF	1
		19	26	28	31	39	2*4
		43	2*45	CONTROL	28	41	4
		45					
SUPPLY	EQU	DECLARED	38	DEFINED	43	IMPL-ASN	4
		REF	47				
TRANSPORT	MODEL	DECLARED	47	DEFINED	47	REF	4
X	VAR	DECLARED	31	IMPL-ASN	49	REF	3
		41	43	45	2*51		
Z	VAR	DECLARED	32	IMPL-ASN	49	REF	4
		49					

For example, the cross-reference list tells us that the symbol A is a parameter that was declared in line 10, defined (assigned value) in line 11, and referenced in line 43. The symbol I has a more complicated entry in the cross-reference list. It is shown

to be a set that was declared and defined in line 5. It is referenced once in lines 10, 19, 26, 28, 31, 38 and 45 and referenced twice in lines 41 and 43. Set I is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 28, 41, 43 and 45.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model due to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For our example, this list is as follows.

```
SETS
I           CANNING PLANTS
J           MARKETS

PARAMETERS
A           CAPACITY OF PLANT I IN CASES
B           DEMAND AT MARKET J IN CASES
C           TRANSPORT COST IN THOUSANDS OF DOLLARS PER CASE
D           DISTANCE IN THOUSANDS OF MILES
F           FREIGHT IN DOLLARS PER CASE PER THOUSAND MILES

VARIABLES
X           SHIPMENT QUANTITIES IN CASES
Z           TOTAL TRANSPORTATION COSTS IN THOUSANDS OF DOLLARS

EQUATIONS
COST        DEFINE OBJECTIVE FUNCTION
DEMAND      SATISFY DEMAND AT MARKET J
SUPPLY      OBSERVE SUPPLY LIMIT AT PLANT I

MODELS
TRANSPORT
```

10.3 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with "****" and contain a "$" directly below the point at which the compiler thinks the error occurs. The $ is followed by a numerical error code which is explained after the echo print. Several examples follow.

Example 1: Entering the statement

```
SET  Q  quarterly time periods  / SPRING, SUM, FALL, WTR / ;
```

results in the echo

```
     1     SET  Q  QUARTERLY TIME PERIODS  / SPRING, SUM, FALL, WTR / ;
****                                                   $160
```

In this case, the GAMS compiler indicates that something is wrong with the set element "SUM." At the bottom of the echo print, we see the interpretation of error code 160:

```
ERROR MESSAGES
160 UNIQUE ELEMENT EXPECTED
```

The problem is that "SUM" is a reserved word meaning "summation," so our set element must have a unique name like "SUMMER." This is a common beginner error. The complete list of reserved words is as follows.

Reserved Words and Symbols

ABORT	EQUATION	MODEL	PARAMETERS	SUM
ACRONYM	EQUATIONS	MODELS	POSITIVE	SYSTEM
ACRONYMS	FREE	NA	PROD	TABLE
ALIAS	GE	NE	SCALAR	USING
ALL	GT	NEGATIVE	SCALARS	VARIABLE
AND	INF	NO	SET	VARIABLES
ASSIGN	INTEGER	NOT	SETS	XOR
BINARY	LE	OPTION	SMAX	YES
CARD	LOOP	OPTIONS	SMIN	
DISPLAY	LT	OR	SOLVE	
EPS	MAXIMIZING	ORD	SOS1	
EQ	MINIMIZING	PARAMETER	SOS2	

Example 2: Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of C(I,J), as follows.

```
PARAMETER C(I,J)  transport cost in thousands of dollars per case
          C(I,J) = F * D(I,J) / 1000 ;
```

Here is the resulting output.

```
   16     PARAMETER C(I,J)  transport cost in thousands of dollars per case
   17               C(I,J) = F * D(I,J) / 1000 ;
****                     $97       $195$96$194$1

          ERROR MESSAGES
            1  REAL NUMBER EXPECTED
           96  BLANK NEEDED BETWEEN IDENTIFIER AND TEXT
                 (-OR- ILLEGAL CHARACTER IN IDENTIFIER)
                 (-OR- CHECK FOR MISSING ';' ON PREVIOUS LINE)
           97  EXPLANATORY TEXT CAN NOT START WITH '$', '=', or '..'
          (-OR- CHECK FOR MISSING ';' ON PREVIOUS LINE)
          194  SYMBOL REDEFINED
          195  SYMBOL REDEFINED WITH A DIFFERENT TYPE
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the others! The first error, code 97, indicates that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16, rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.

Unfortunately, you cannot always expect the error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the C entry in the cross-reference list and noticing that it was never assigned.

```
          SYMBOL      TYPE   REFERENCES
          C           PARAM  DECLARED       15      REF       17
```

Example 3: Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with "Seattle" spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```
     4    SETS
     5         I   canning plants   / SEATTLE, SAN-DIEGO /
     6         J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;
     7
     8    TABLE D(I,J)  distance in thousands of miles
     9                        NEW-YORK       CHICAGO       TOPEKA
    10         SEATLE             2.5           1.7           1.8
****              $170
    11         SAN-DIEGO          2.5           1.8           1.4  ;

          ERROR MESSAGES
          170  DOMAIN VIOLATION FOR ELEMENT
```

Example 4: Similarly, if we mistakenly enter DEM(J) instead of B(J) as the right-hand-side of the demand constraint, the result is:

```
   45    DEMAND(J) ..   SUM(I, X(I,J))  =G=  DEM(J) ;
****                                          $140

          ERROR MESSAGES
          140  UNKNOWN SYMBOL, ENTERED AS PARAMETER
```

Example 5: The next example is a mathematical error, which is sometimes committed by novice modelers, and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

$$\text{For all } i, \quad \Sigma_i \, x_{ij} = 100.$$

There are two errors in this equation, both having to do with the control of indices. Index i is over-controlled and index j is under-controlled.

You should see that index i is getting conflicting orders. By appearing in the quantifier, "for all i," it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index j is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

```
          MEANINGLSS(I) ..   SUM(I, X(I,J))  =E=  100 ;
****                              $125   $149

ERROR MESSAGES
125  SET IS UNDER CONTROL ALREADY              [This refers to set I.]
149  UNCONTROLLED SET ENTERED AS CONSTANT      [This refers to set J.]
```

A great deal more information about error reporting is given in Chapter ##. Well designed error messages are a big help in getting models implemented quickly and correctly.

10.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question. A product of the SOLVE command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

```
DEMAND(J) ..   SUM(I, X(I,J))  =G=  B(J) ;
```

while the equation listing of specific constraints is

```
---- DEMAND       =G=  SATISFY DEMAND AT MARKET J

DEMAND(NEW-YORK)..  X(SEATTLE,NEW-YORK) + X(SAN-DIEGO,NEW-YORK) =G= 325 ;

DEMAND(CHICAGO)..  X(SEATTLE,CHICAGO) + X(SAN-DIEGO,CHICAGO) =G= 300 ;

DEMAND(TOPEKA)..  X(SEATTLE,TOPEKA) + X(SAN-DIEGO,TOPEKA) =G= 275 ;
```

The default output is three specific equations (if there are at least that many) for each generic equation. To change the default, insert an input statement prior to the SOLVE statement:

```
OPTION LIMROW = r ;
```

where r is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the previous command can be appended:

```
OPTION LIMROW = r, LIMCOL = c ;
```

where c is the desired number of columns. (Setting LIMROW = 0 and LIMCOL = 0 is a good way to save paper after your model is debugged.)

In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

10.5 Model Statistics

The last section of output that GAMS produces before invoking the solver is statistics about the model's size, as shown below for the transportation example.

```
MODEL STATISTICS

BLOCKS OF EQUATIONS     3       SINGLE EQUATIONS     6
BLOCKS OF VARIABLES     2       SINGLE VARIABLES     7
NON ZERO ELEMENTS      19
```

The "BLOCK" counts refer to the number of generic equations and variables. The "SINGLE" counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to characterize the degree of nonlinearity in the problem.

10.6 Status Reports

After the solver executes, GAMS prints out a brief "SOLVE SUMMARY," whose two most important entries are "SOLVER STATUS" and the "MODEL STATUS." For our transportation problem the solve summary is as follows:

```
                S O L V E      S U M M A R Y

     MODEL   TRANSPORT           OBJECTIVE  Z
     TYPE    LP                  DIRECTION  MINIMIZE
     SOLVER  BDMLP               FROM LINE  49

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE               153.6750

     RESOURCE USAGE, LIMIT          0.050     1000.000
     ITERATION COUNT, LIMIT         4         1000
```

The status reports are preceded by the same "****" string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you peruse an output file for the first time. The desired solver status is "1 NORMAL COMPLETION"

but there are five other possibilities, documented in Chapter ##, which relate to various types of errors and mishaps.

There are 11 possible model statuses, including the usual linear programming termination states ("1 OPTIMAL," "3 UNBOUNDED," "4 INFEASIBLE"), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is "2 LOCALLY OPTIMAL." The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality.

In integer programming, the status to look for is "10 INTEGER SOLUTION." This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

10.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in a standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit and marginal. The row output is grouped by generic equation block and the column output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for SUPPLY(I), DEMAND(J) and X(I,J) are as follows:

```
---- EQU SUPPLY       OBSERVE SUPPLY LIMIT AT PLANT I
                LOWER      LEVEL      UPPER   MARGINAL
SEATTLE          -INF    350.000    350.000        EPS
SAN-DIEGO        -INF    550.000    600.000          .


---- EQU DEMAND       SATISFY DEMAND AT MARKET J
                LOWER      LEVEL      UPPER   MARGINAL
NEW-YORK      325.000    325.000       +INF      0.225
CHICAGO       300.000    300.000       +INF      0.153
TOPEKA        275.000    275.000       +INF      0.126
```

```
---- VAR X                 SHIPMENT QUANTITIES IN CASES
                           LOWER      LEVEL      UPPER     MARGINAL
SEATTLE   .NEW-YORK          .        50.000      +INF         .
SEATTLE   .CHICAGO           .       300.000      +INF         .
SEATTLE   .TOPEKA            .          .         +INF       0.036
SAN-DIEGO.NEW-YORK           .       275.000      +INF         .
SAN-DIEGO.CHICAGO            .          .         +INF       0.009
SAN-DIEGO.TOPEKA             .       275.000      +INF         .
```

The single dots "." in the output represent zeroes. The entry "EPS," which stands for "epsilon," means very small but nonzero. In this case, EPS indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with EPS rather than zero to facilitate restarting the optimizer from the old basis.)

If the solver's results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with "INFES" or "NOPT," respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked "UNBND."

At the end of the solver's solution report is a very important "report summary," which gives a tally of the total number of nonoptimal, infeasible and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

```
**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
```

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the .L and .M fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed and GAMS automatically formats and labels an appropriate array. For example, the input statement

```
DISPLAY X.L, X.M ;
```

results in the following output.

```
----     51 VARIABLE  X.L              SHIPMENT QUANTITIES IN CASES
               NEW-YORK      CHICAGO       TOPEKA
SEATTLE          50.000      300.000
SAN-DIEGO       275.000                    275.000
```

```
----     51 VARIABLE  X.M            SHIPMENT QUANTITIES IN CASES
                CHICAGO      TOPEKA
SEATTLE                       0.036
SAN-DIEGO         0.009
```

As seen in reference maps, equation listings, solution reports and optional displays, GAMS saves the documentary text and "parrots" it back throughout the output to help keep the model well documented.

Conclusions

This tutorial has demonstrated several of the features of GAMS's design which enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS, versus a matrix generator or conversational solver[5].

1. By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.

2. Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.

3. You save time and reduce generation errors by creating whole sets of closely related constraints in one GAMS statement.

4. You can save time and reduce input errors by providing formulae for calculating the data, rather than entering them explicitly.

5. The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.

6. The output of GAMS is easy to read and use. The solver's solution report is automatically reformatted by GAMS

---

[5] For information concerning other algebraic modeling languages that are available or in development, see Fourer [1983], Fourer, Gay and Kernighan [1987], Geoffrion [1987], Schrage [1987] and Welch [1987].

so that related equations and variables are grouped together and appropriately labeled. Also, the DISPLAY command allows you to modify and tabulate results very easily.

7. If you are teaching or learning modeling, you can benefit from the GAMS compiler's insistence that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which GAMS catches errors should greatly reduce development time.

8. By using the "dollar" operator and other advanced GAMS features not covered in this tutorial, you could efficiently implement large-scale models. Specific applications of the dollar operator include the following:

a. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem down in the range of solvability.

b. It can be used to build complex summations and products, which can then be used in equations or customized reports.

c. It can be used for issuing warning messages or for terminating prematurely, conditioned upon context-specific data edits.

References

J. Bisschop and A. Meeraus, "Selected Aspects of a General Algebraic Modeling Language," in **Optimization Techniques: Proceedings of the 9th IFIP Conference on Optimization Techniques**, Part 2, K. Iracki, K. Malanowski and S. Walukiewicz (eds.), Springer-Verlag, Berlin, 223-233, 1980.

J. Bisschop and A. Meeraus, "Toward Successful Modeling Applications in a Strategic Planning Environment," in **Large Scale Linear Programming**, G.B. Dantzig, M.A.H. Dempster and M.J. Kallio (eds.), International Institute for Applied Systems Analysis, Laxenburg, Austria, 711-745, 1981.

A. Brooke, D. Kendrick, and A. Meeraus, **GAMS: A User's Guide**, The Scientific Press, Redwood City, CA, forthcoming in 1988.

G.B. Dantzig, **Linear Programming and Extensions**, Princeton University Press, Princeton, NJ, 1963.

R. Fourer, "Modeling Languages versus Matrix Generators for Linear Programming," **ACM Transactions on Mathematical Software 9**, 143-183, 1983.

R. Fourer, D. Gay and B. Kernighan, "AMPL: A Mathematical Programming Language," Computing Science Technical Report No. 133, AT&T Bell Laboratories, Murray Hill, NJ, 1987.

A.M. Geoffrion, "An Introduction to Structured Modeling," **Management Science** 33, 547-588, 1987.

L. Schrage, "The LINGO Modeling Language," presented at ORSA/TIMS Joint National Meeting, St. Louis, MO, 1987.

J.S. Welch, Jr., "PAM -- A Practitioner's Guide to Modeling," **Management Science** 33, 611-625, 1987.

DISTRIBUTION LIST

	NO. OF COPIES
Library (Code 0142) Naval Postgraduate School Monterey, CA 93943-5000	2
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Office of Research Administration (Code 012) Naval Postgraduate School Monterey, CA 93943-5000	1
Center for Naval Analyses 4401 Ford Avenue Alexandria, VA 22302-0268	1
Library (Code 55) Naval Postgraduate School Monterey, CA 93943-5000	1
Operations Research Center, Rm E40-164 Massachusetts Institute of Technology Attn: R. C. Larson and J. F. Shapiro Cambridge, MA 02139	1
Koh Peng Kong OA Branch, DSO Ministry of Defense Blk 29 Middlesex Road SINGAPORE 1024	1
Arthur P. Hurter, Jr. Professor and Chairman Dept of Industrial Engineering and Management Sciences Northwestern University Evanston, IL 60201-9990	1
Institute for Defense Analysis 1800 North Beauregard Alexandria, VA 22311	1

END

DATE

FILMED

APRIL

1988

DTIC