COMPUTATIONAL ECONOMICS

Transportation in GAMS

David A. Kendrick
Ruben P. Mercado
Hans M. Amman

1 Introduction

The transportation problem was made famous among economists by the work of Tjalling Koopmans (1951) and of Robert Dorfman, Paul Samuelson, and Robert Solow (1958), several of whom won the Noble Prize in economics. This kind of model is a most natural way to pose the problem of finding the most efficient place and manner of producing and shipping goods. The model posits supplies of a good at a number of plants and demands for that good in a number of markets and seeks to find the amount that each plant should ship to each market in order to minimize transportation costs. The transportation model is also the foundation for much more elaborate linear programming industrial models, such as those for steel, oil, aluminum, fertilizer, and computers. These models focus not only on transportation but also on production and investment.

We begin with a mathematical representation of the transportation problem and then move to a discussion of how this model can be represented in the GAMS software.

2 Mathematical Representation

As an example [adapted from Dantzig (1963)] for this chapter we use the fishing industry with canneries in Seattle and San Diego and markets in New York, Chicago, and Topeka (Kansas). In this model we try to find the pattern of shipments from the canneries to markets that have the lowest transportation cost while satisfying the fixed demand at the markets and without shipping more from any cannery than its capacity.

The model is stated mathematically as follows:

For the sets

$$\begin{aligned} I \text{ plants} &= \text{Seattle, San Diego} \\ J \text{ markets} &= \text{New York, Chicago, Topeka} \end{aligned}$$

find the

$$x_{ij} \text{ from plant } i \text{ to market } j \tag{1}$$

to minimize the transportation cost

$$z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{2}$$

where

$$c_{ij} = \text{transportation cost from plant } i \text{ to market } j \text{ per unit shipped}$$

The criterion function (3) is minimized subject to the constraints that no more be shipped from each plant than its capacity

$$\sum_{j \in J} x_{ij} \leq a_i \quad i \in I \tag{3}$$

where

$$a_i = \text{the capacity of plant } i$$

and that no less be shipped to each market than its demand

$$\sum_{i \in I} x_{ij} \geq b_j \quad j \in J \tag{4}$$

where

$$b_j = \text{the demand at market } j \tag{5}$$

while requiring that all the shipments be nonnegative:

$$x_{ij} \geq 0 \qquad i \in I \qquad j \in J \tag{6}$$

Next we turn to the representation of this model in GAMS.

3 GAMS Representation

GAMS (General Algebraic Modeling System) was developed in the 1970s at the World Bank by Alexander Meeraus and his colleagues. The user's guide for this system is by Brooke, Kendrick, Meeraus, and Raman (1998). GAMS was designed as a set driven high-level language that would facilitate the development of linear and nonlinear programming models of industry, agriculture, and finance. Thus it is not necessary to write a separate equation for each commodity, time period, crop, or equity, but rather only to create equations and variables indexed over sets of commodities, time periods, crops, equities, and so forth. In this way a model with thousands of equations can be represented in a GAMS statement with only a few set specifications, variables, and equations—all of which might fit on a single page. This not only decreases a tedious, labor-intensive part of model development but also substantially reduces the likelihood of errors in the model specification.

GAMS has also become widely used because of the ability to represent in it any model that can be expressed in algebra. In particular there are now many computable general equilibrium, agricultural, and financial models in GAMS, as well as a wide variety of other types of economic models. For a listing of several hundred GAMS models see the GAMS library that comes with the software or access the library at `http://www.gams.com`. These models can be downloaded and solved with the GAMS software.

Many of you will run GAMS on your home computers or in a university computer laboratory. The instructions for fetching the input file and running the program on a personal computer are contained in Appendix A.

The GAMS program corresponding to the transportation problem is available at the book web site under the name trnsport.gms as well as in the GAMS library under the same name. (Note that *transport* is misspelled in this filename.) An extended tutorial on this model is available in the GAMS User's Guide [Brooke et al. (1998)].

The GAMS language uses a syntax that is reasonably close to mathematics. For example, the criterion function for the transportation model is written in mathematics as

$$z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{7}$$

and in GAMS as

```
cost..   z =e= sum((i,j), c(i,j) * x(i,j)) ;
```

In mathematics equations are usually numbered, whereas in GAMS they are named; thus equation (7) gets the name cost and the two dots after cost tell the GAMS compiler that the name has been completed and the equation is about to begin. GAMS also has an unusual way of representing an equal sign, namely =e=. The reason for this is that GAMS also includes less-than-or-equal signs and greater-than-or-equal signs, which are discussed later.

The GAMS language does not make a distinction between set names such as I and the indices of the elements that belong to the sets such as i. This results partly from the fact that GAMS, unlike most programming languages, does not distinguish between upper and lower case letters. Thus one could imagine writing the right-hand side of the foregoing equation in GAMS as

```
  sum((I,J), c(i,j) * x(i,j)) ;
```

to indicate that the sum is over the sets I and J, while the parameter c and the variable x are defined with the subscripts *(i,j)*. However, that is not necessary in GAMS and the user learns to read symbols such as (i,j) as sometimes representing set names and other times representing indices.

Finally, in mathematics the simple juxtaposition of two symbols such as c and x indicates that they are multiplied by one another, whereas in GAMS, as in most other programming languages, it is necessary to explicitly indicate multiplication with an asterisk, *.

Similarly, the capacity constraint is written in mathematics as

$$\sum_{j \in J} x_{ij} \leq a_i \; i \in I \tag{8}$$

and in GAMS as

```
 supply(i) ..  sum (j, x(i,j)) =l=  a(i) ;
```

Thus gets the name supply and the (i) that follows it indicates that there is one constraint of this type for each element in the set I, that is, for each plant. Thus the (i) next to the equation name in GAMS plays the same role as the symbols $i \in I$ play in the mathematics. Also note that the less-than-or-equal-to sign, $\leq$, in mathematics becomes =l=, where the l here indicates the letter l and not the number 1.

The transportation model as stated above could be for a model with two plants and three markets or for one with 50 plants and 200 markets, as we have not yet specified the sets I and J nor the parameters a, b, and c. This is one of the powers of the GAMS language, that is, one can write a model prototype that can be used for any of a number of industries and then specialize it in the set specifications and the parameter definitions for a particular industry in a given country.

We consider next how the sets are specified in the GAMS language. In this model there are two plants and three markets. The set of plants is specified in mathematics as

$$I = \{\text{Seattle, San Diego}\} \tag{9}$$

The equivalent GAMS statement is

```
i = / seattle, san-diego /
```

GAMS uses forward slashes as set delimiters, whereas mathematics use braces.

Once the sets are specified then the data can be input using the parameter and table keywords as shown below. Consider first the use of the parameter keyword to input the capacity and demand data:

```
Parameter
          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  / ;
```

Observe that the parameter a is followed by the set over which it is defined, that is, it is written as a(i). As noted earlier, it is not necessary to include the set here; however, it is a useful precaution because when the set is provided the GAMS complier can check to be sure that all the element names used in the input of the parameter do indeed belong to that set.

Next consider the input of the distance data with the statement

```
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  ;
```

Here the table keyword is used to input the matrix of transportation distances between the markets and the plants.

Next consider the scalar keyword that can be used to input a scalar quantity, in this case f, which is freight cost per case per 1000 miles:

```
Scalar f freight in dollars per case per thousand miles  /90/ ;\\
```

This scalar in turn can be used in a parameter statement to compute the transport cost per case between each plant and market:

```
Parameter c(i,j) transport cost in thousands of dollars per case ;

           c(i,j) = f * d(i,j) / 1000 ;
```

Note here that the new element `c(i,j)` is first declared with a parameter statement and then defined with a mathematical statement in which `f` is multiplied by `d` and divided by 1000. Here you see that the parameter keyword in GAMS is much more versatile than just being a way to input vectors.

The foregoing computation of the `c(i,j)` parameter illustrates one theme in the use of the GAMS language. The user is encouraged to enter the raw data for the model in the GAMS statement and to show explicitly all the mathematical transformations that are performed on that data before it becomes part of the model equations.

Consider a word of warning about the data for the transportation model from the GAMS library. Note that the distances in Table `d(i,j)` are listed as the same from Seattle to New York and from San Diego to New York, that is, 2500 miles. This can cause multiple optimal solutions to the model, which can be confusing. Therefore, when first using this model it is probably wise to make these distances different. For example, one might change the number for the Seattle to New York distance from 2.5 (thousand miles) to 2.7 (thousand miles).

Next consider the variables and equations part of the GAMS representation

```
      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) ;
```

The keyword Variables is used to declare the variables and in the process one indicates the sets over which the variables are defined. For example, the variable x is defined for the set of plants I and the set of markets J, so it is listed as `x(i,j)`. The restriction that the shipment variables must be nonnegative as shown in equation (6) is carried in the `Positive Variable x` statement.

The names of the equations are listed after the Equations keyword along with the sets over which they are defined. For example, there is a supply equation for each plant so that equation is defined as supply(i).

The final statements in the GAMS specification are:

```
Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;
```

The Model keyword is used to give the model a name—in this case transport—and to indicate the equations that are included in the model. One may either list a subset of the equation names here or if the model consists of all the foregoing equations then the all keyword can be used. The model is then solved with the Solve, using, and minimizing keywords. From a mathematical point of view, the transportation problem is a particular case of what is known as *linear programming*, that is, a problem in which one seeks to optimize a linear objective function subject to a set of linear constraints. See Appendix B for an introduction to linear programming. The `lp` in the solve statement tells GAMS to use its linear programming solver to compute the solution to the model and the `z` is the criterion value that is to be minimized. Since the model contains indexed equations, GAMS uses a stacking method as discussed in Appendix C.

Finally, the display statement requests that the activity levels for the shipment variables, that is, `x.l` and the marginal values `x.m` for these same variables be displayed in tables. Learning all this syntax for a programming language may seem complicated at first. However, the structure of the model helps to simplify things. Note in the complete GAMS statement of the model that follows this paragraph that the model is defined in steps:

- first the sets
- then the parameters
- then the variables
- then the equations
- finally the model and solve statements.

It takes awhile to adjust to all the details but the overall structure and form of a GAMS representation of a model can be grasped quickly. The entire GAMS listing of the model follows:

```
$Title  A Transportation Problem (TRNSPORT,SEQ=1)

 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 ;
```

This completes the discussion of the input for the model. Next we turn to the way to solve the model and a discussion of the results.

4 Results

As we noted earlier, Appendix A contains instructions on how to access the *.gms file from the GAMS library, how to solve the model, and how to examine the results by using the listing file,*.lst. This last step can seem complicated at first because the GAMS output files contain a substantial amount of information about the structure of the model and its solution. However, it is simple enough to jump around in the file to examine the key parts.

One should first locate the Solve Summary part of the output. To do this, search in the editor for the string SOLVER STATUS. When you do so you see a section of the output that looks like

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

      MODEL   TRANSPORT           OBJECTIVE  Z

      TYPE    LP                  DIRECTION  MINIMIZE

      SOLVER  BDMLP               FROM LINE  70

 **** SOLVER STATUS     1 NORMAL COMPLETION

 **** MODEL STATUS      1 OPTIMAL

 **** OBJECTIVE VALUE              153.6750
```

Every time you solve a GAMS model you should check this section of the output to be sure that the model was solved successfully. The words NORMAL COMPLETION here indicate that that is the case. If the solution procedure was not successful you find words like INFEASIBLE or UNBOUNDED. Be on guard against the fact that the GAMS output provides a solution to the model even when that solution is infeasible. However, the solution provided is not the optimal solution but rather the last one tried before it was determined that the solution was infeasible. For this reason, it is particularly important to check the SOLVER STATUS and MODEL STATUS after each run and before the solution variables are used.

Next skip down the output across the sections labeled —- EQU until you get to the sections labeled —- VAR, which look like

```
---- 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 interesting part here is the activity level of the shipment variables x in the column labeled LEVEL, which shows, among other things, that 50 cases were shipped from Seattle to New York and 300 cases were shipped from Seattle to Chicago. This is the solution of the model that we were looking for. These same results are shown a little further down in the output in a section labeled VARIABLE X.L, which is the result of the display statement in the GAMS input. That output is as follows:

```
----     72 VARIABLE  X.L           shipment quantities in cases

             NEW-YORK     CHICAGO      TOPEKA

SEATTLE        50.000     300.000

SAN-DIEGO     275.000                 275.000
```

This table is somewhat easier to read than the default output and thus you can see the reason that most GAMS input files end with a series of display statements. These tables are easily found since they are at the end of the long GAMS output so the user can quickly scroll to the bottom of the file and find the key results. However, they will be there only if you remember to add a display statement at the end of the GAMS input statement.

There is just one other key piece of the GAMS output file that we should look at before we turn our attention elsewhere. That piece is in the —- EQU section that we skipped over earlier; you can find it quickly by scrolling back up to it or by searching for it with the editor. The part of interest is the equation-wise output for the demand constraints, which looks like the following:

```
---- 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
```

In this case we are interested in the MARGINAL column. These values are called *shadow prices* or *dual* variables and have important economic meaning. They show us that for each additional unit of demand in New York the objective function has to increase by 0.225 but by only 0.153 at Chicago and 0.126 at Topeka. These numbers are like prices and indicate that it is substantially more expensive to supply fish to New York than to Chicago or Topeka. Similar numbers in electric power models can be used by regulators to determine the price of power in cities that are nearby or far away from electric power generation facilities such as dams or nuclear or coal burning plants.

In summary, when looking at the GAMS output you should first check to be sure that the problem was solved satisfactorily. Then focus on the variables section and finally take a look at the equation section.

5 Experiments

As a simple experiment, you might first change the number for the distance from Seattle to New York from 2.5 (thousand miles) to 2.7 (thousand miles)—to eliminate the multiple-solution problem discussed earlier—and then solve the model again. Or you might decrease the demand at one or more markets or increase the supply in one or more plants in order to analyze the effects on the optimal solution. However, when changing the supply and demand parameters you must be careful to ensure that the total supply is greater than or equal to the total demand—otherwise the solution to the model is infeasible.

A more complicated experiment is to add additional markets and/or plants. This helps one to learn quickly how the sets are specified and the ripple effect this has on required changes in the parameter and table statements. In the process one may switch the model from a focus on fish to steel or fertilizer or glass or computers or whatever industry is of interest.

A further complication would be the addition of the production cost at each plant. This could be done by introducing a new parameter as follows:

```
        prodcost(i)  production cost of plant i per case

          /    seattle     15

               san-diego   18  / ;
```

Then the criterion function would also have to be changed from

```
       cost ..     z  =e=  sum((i,j), c(i,j)*x(i,j)) ;\\
```

to

```
       cost .. z  =e=  sum((i,j), (c(i,j) + prodcost(i))*x(i,j) );
```

Then the model can be used to analyze the effects of production differences at plants as well as transportation cost differences between pairs of plants and markets.

If one wants to change the criterion from cost minimization to profit maximization it can be done by introducing prices in each market. One way to do this is to approximate a nonlinear demand function with a piecewise linear function. For example, the demand might be thought of in three segments, with the total demand in market *j* being equal to the sales in the three segments, namely

$$s = s_1 + s_2 + s_3$$

The subscript *j* for the market is omitted here in order to simplify the notation in the following development. The revenue generated by sales at this one market could then be written

$$rev = p_1 s_1 + p_2 s_2 + p_3 s_3$$

while being careful to ensure that the parameter for the price in the first segment is higher than for the price in the second segment, which is, in turn, higher than for the price in the third segment, that is,

$$p_1 > p_2 > p_3$$

and put an upper bound on sales in each of the first two segments, that is,

$$s_1 \leq \beta_1$$
$$s_2 \leq \beta_2$$

where β_1 is the upper bound in segment 1 and $\beta_2$is the upper bound in segment 2.

Then the criterion value becomes the maximization of profit, which is revenue minus cost, that is,

$\pi = \text{rev} - \text{cost}$

where

$$rev = \sum_{j \varepsilon J} \left(p_{1j} s_{1j} + p_{2j} s_{2j} + p_{3j} s_{3j} \right)$$

with

$$\begin{aligned} p_{kj} &= \text{price in the } k \text{ segment in market } j \\ s_{kj} &= \text{sales in the } k \text{ segment in market } j \end{aligned} \tag{10}$$

and the cost includes both the transportation and the production costs.

6 Further Reading

In the 1970s and 1980s there was a project at the World Bank, under the leadership of Hollis Chenery and more directly of Ardy Stoutjesdijk, that focused on the development of a variety of industrial models for steel, fertilizer, pulp and paper, and other industries. These models used the GAMS language, which was under development there at that time by Alexander Meeraus and his colleagues. One of those models, namely the one on the Mexican steel industry, is a logical follow-on to the model developed in this chapter, namely Kendrick, Meeraus, and Alatorre (1984). There is also a shorter, more intuitive, chapter on this model in Kendrick (1990), and various versions of the model itself are available in the GAMS library.

References

Chiang, A. C.: 1974, *Fundamental methods of mathematical economics*, International Student Edition, 2nd edn, McGraw-Hill Kogakusha, Tokyo, Japan.

Kendrick, D. A.: 1990, *Models for Analyzing Comparative Advantage*, Kluwer Academic Publishers, Dordrecht, the Netherlands.

Kendrick, D. A., Meeraus, A. and Alatorre, J.: 1984, *The Planning of Investment Programs in the Steel Industry*, Johns Hopkins University Press, Baltimore, Maryland.

Appendix

A Running GAMS

Appendix A provides the details for running the GAMS software on a PC. In order to use GAMS with other input files, substitute the appropriate file name for trnsport.gms in the following. For help and information about obtaining GAMS go to the GAMS Development Corporation web site at

`http://www.gams.com`

There is a student version of the software that can be downloaded and that can solve all or almost all of the models used in this book. It the model is too large, usually a small change in the number of time periods or some other set is sufficient to reduce the size so that it runs on the student version.

- Go to the book web site at

  `http://www.eco.utexas.edu/compeco`

  and to the *Input Files for Chapters in the Book* section of the web site. Right click on the `trnsport.gms` file name and select the `Save Target As` ... option in order to save the file in your preferred directory.

- Choose Programs from the Start menu and then choose GAMS and gamside. Choose Open from the File menu, navigate to the trnsport.gms file, and open it for editing. Note in the complete GAMS statement of the model that, as is the usual case in GAMS, the model is defined in steps:

  1. first the sets
  2. then the parameters
  3. then the variables
  4. then the equations
  5. and finally the model and solve statements.

- Solve the model by choosing Run from the File menu and then check the solution log to be sure that you have

```
SOLVER STATUS: 1 NORMAL COMPLETION
```

and

```
MODEL STATUS: 1 OPTIMAL
```

Then close the log file window.

- Click on the trnsport.lst file window and scroll through this listing file to see the solution. Note that the `*.lst` file extension used here is an abbreviation for a *listing* of the output file. Note that the GAMS output has the following structure:

  Echo Print shows a listing of the input file with the line numbers added.

  Error Messages In the case of errors in the input file, they will be signaled by GAMS with `****` on the leftmost part of the corresponding line of input where the error was found, and with `$number` just below the part of the line of input where the error is located, where `number` contains a specific error code. Then, at the end of the list of the input file, GAMS displays the explanation of each of the error codes found.

  `Equation Listing` Shows each equation of the model with the corresponding values for sets, scalars, and parameters.

  `Column Listing` Shows a list of the equations' individual coefficients classified by columns.

  `Model Statistics` Shows information such as model number of equations, number of variables, and so on.

  `Solve Summary` Shows information such as solver and model status at the end of the GAMS run, and so on.

  `Solution Listing` Shows the solution values for each equation and variable in the model. Each solution value is listed with four pieces of information, where a dot `.` means a value of zero, `EPS` a value near zero, and `+INF` and `-INF` mean plus or minus infinite, respectively:

  ```
  LOWER (the lower bound)

  LEVEL (the solution level value)

  UPPER (the upper bound)
  ```

`MARGINAL` (the solution marginal value; for linear or nonlinear programming problems, it corresponds to the increase in the objective value owing to a unit increase in the corresponding constraint)

`Report Summary` Shows the count of rows or columns that are infeasible, nonoptimal, or unbounded.

B Linear Programming Solvers

A linear programming problem is one of maximizing a linear objective function subject to a set of linear constraints. In economics, it is also frequently required that the variables of the problems be nonnegative. Thus, in mathematical terms a linear programming problem can be expressed as

$$\max y = b'x \tag{B-1}$$

subject to

$$\begin{aligned} Ax &\leqslant k \qquad \text{(B-2)} \\ x &\geqslant 0 \qquad \text{(B-3)} \end{aligned}$$

where y is a scalar, x is a vector of variables, b and k are vectors of constants, and A is a matrix. If the problem is one of minimization, it can be written as one of maximizing the objective function with a negative sign and changing the direction of the inequalities by multiplying both sides by –1. For an intuitive graphical representation, suppose that we have a problem with two variables and three restrictions, that is,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \tag{B-4}$$

and

$$k = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} \tag{B-5}$$

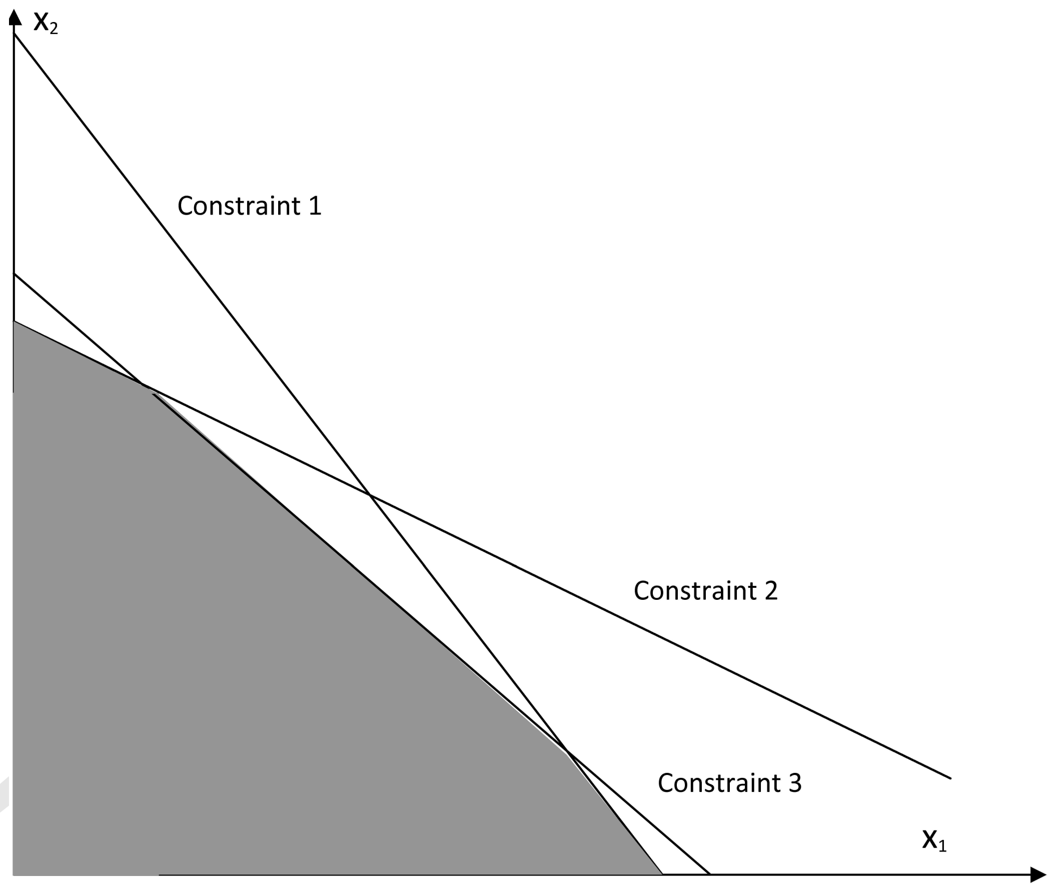
which can be represented as in Figure 1

We can see that the problem constraints define an area—the shaded one—that contains all the feasible solutions. It is a closed, convex, and lower-bounded set, also known as a simplex. In Figure 2 we added the corresponding level curves of the objective function. Since for this example

$$y = b_1x_1 + b_2x_2 \tag{B-6}$$

those level curves are given by

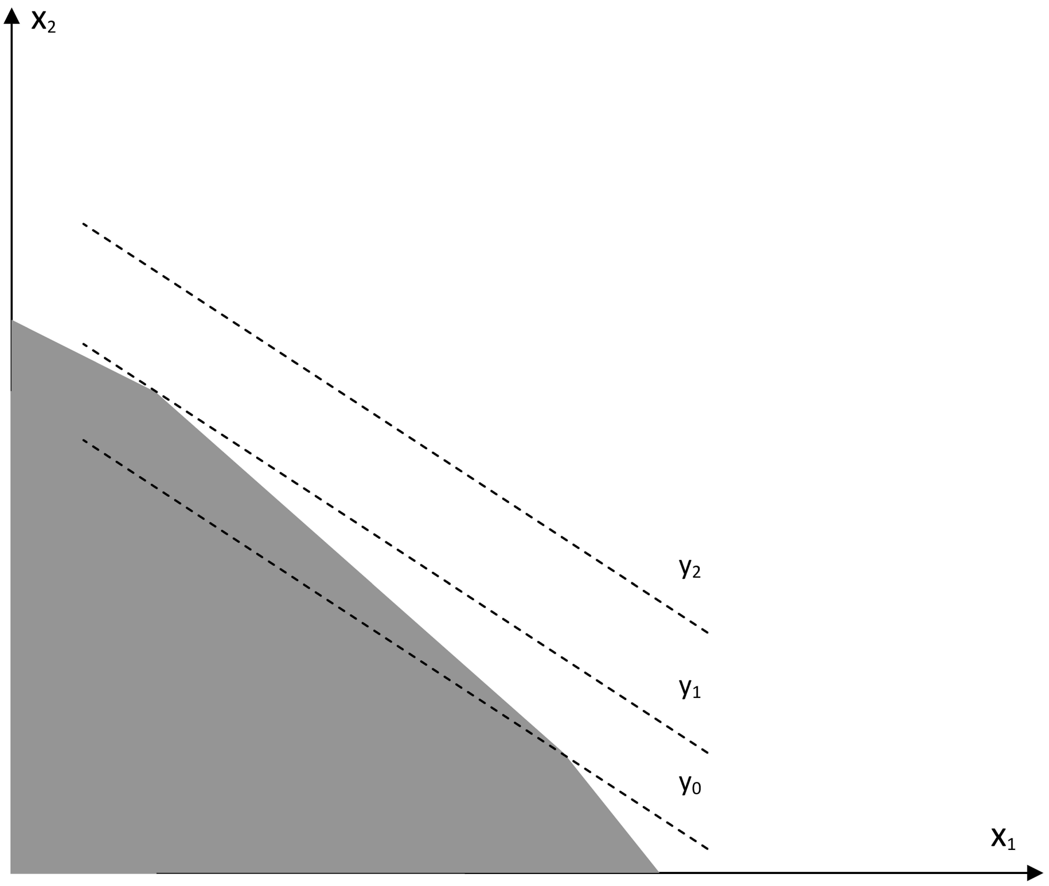
Figure 1: Feasible solution set.



$$x_2 = \frac{y}{b_2} - \frac{b_1}{b_2} x_1 \tag{B-7}$$

We have one level curve for each value of y.

Figure 2: Level curves



We can see that the maximum feasible y is y_1. Generalizing, we can say that the optimum value of a linear programming problem is obtained at the point at which a level curve is tangent to the simplex of feasible solutions, and this always happens at a vertex of the simplex. Note also that multiple—actually an infinite number—of solutions are obtained when the level curve is tangent to a segment between two vertices.

Thus, a solution method could be one that focuses on the evaluation of the vertices of the simplex of feasible solutions. A rudimentary method would evaluate all vertices and choose the one that generates the highest value—for a maximization problem—of the objective function. However, the number of vertices grows very quickly as the number of variables and constraints increases.

A more efficient method, used by the default GAMS solver BDMLP, is the iterative procedure known as the simplex method. Starting from a given vertex, this method looks for the best direction of motion toward another vertex. To do so, it starts by transforming the inequality restrictions into equalities by adding new nonnegative variables known as *slack variables*. In our two-variable–three-restriction example, this is equivalent to writing the new constraints as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + x_3 = k_1 \\ a_{21}x_1 + a_{22}x_2 + x_4 = k_2 \\ a_{31}x_1 + a_{32}x_2 + x_5 = k_3 \end{aligned} \tag{B-8}$$

or, in matrix notation,

$$[A\ I]\ x = k \tag{B-9}$$

where I is a 3×3 identity matrix and the vector x is now

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{B-10}$$

Note that the new matrix $[A\ I]$ is a 3×5 matrix. Thus, if we set to zero any two variables in x we are left with a 3×3 matrix and a 3×3 system of linear equations. This system has a solution if the corresponding row (column) vectors in the matrix are linearly independent. Moreover, that solution, which corresponds to the base of the tridimensional space spanned by those vectors, is a vertex of the simplex of feasible solutions. Thus, we name it the *basic feasible solution*.

The next step in the simplex method is to evaluate the solution to check whether we are at the optimum. To do so, we compute

$$\frac{\partial y}{\partial x_{NB}} \tag{B-11}$$

where x_{NB} are the nonbasic variables. If any one of these derivatives is greater than zero, we are not at the optimum since the objective function could be incremented by increasing the corresponding nonbasic variable. The next step is to move to another vertex incorporating this variable into the base and deleting one of the variables previously in the base. The selection of the basic variable to be deleted is more involved. Ideally, we should delete the variable that most constrains the potential increase in the objective expected from incorporation of the new basic variable. To do so, the constraints have to be rewritten, now with the basic variables as functions of the nonbasic ones, and the resulting system has to be analyzed.

We then continue evaluating the objective function and incorporating/deleting variables to the basic solution until we reach an optimum.

For more detailed presentations of the simplex method, refer to Chiang (1974), Rardin (1998), and Silverberg and Suen (2001). For details on GAMS linear programming solvers, see the corresponding GAMS Solvers manuals at http://www.gams.com.

C The Stacking Method in GAMS

As a compact way of expressing a multiequation model, GAMS allows us to write indexed equations. As seen in several of the chapters in this book, those indices may represent commodities, locations, time periods, and so on.

For example, the equations corresponding to a problem such as

$$\max J = \sum_{i=0}^{2} w_1 x_i + w_2 y_i \tag{C-1}$$

subject to the constraints

$$a_{11} x_i + a_{12} y_i = b_1 \tag{C-2}$$
$$a_{21} x_i + a_{22} y_i = b_2 \tag{C-3}$$

can be represented in GAMS as

```
eqj..        j =e= sum(i, w1 * x(i) + w2 * y(i));
eq1(i)..     a11 * x(i) + a12 * y(i)) =e= b1;
eq2(i)..     a21 * x(i) + a22 * y(i)) =e=  b2;
```

When the index set is $i = \{0, 1, 2\}$ the model is expanded and stacked in the following way:

```
j =e= w1*x(0) + w2*y(0) + w1*x(1) + w2*y(1) + w1*x(2) + w2*y(2)
eq1(0)..     a11 * x(0) + a12 * y(0)) =e= b1;
eq2(0)..     a21 * x(0) + a22 * y(0)) =e= b2;
eq1(1)..     a11 * x(1) + a12 * y(1)) =e= b1;
eq2(1)..     a21 * x(1) + a22 * y(1)) =e= b2;
eq1(2)..     a11 * x(2) + a12 * y(2)) =e= b1;
eq2(2)..     a21 * x(2) + a22 * y(2)) =e= b2;
```

Note that previously we had a model with an objective function and two indexed equations and two variables [x(i) and y(i)] and now we have a model with one objective function, six equations, and six variables [x(0), x(refeq1), x(refeq2), y(0), y(refeq1) and y(refeq2)]. Thus, before solving the model, GAMS transforms a model of *n* indexed equations into one of *n x card* equations plus the objective function, where *card* indicates the number of elements in the index set. If the index denotes time periods, this is equivalent to transforming a dynamic model with *n* indexed equations and *t* time periods into an equivalent static model of $n \times t$ equations plus the objective function.

When, as in Chapters 8 and 13, we are interested in solving a system of equations and not an optimization problem, we just set the objective function equal to any constant value (i.e., j =e= 0;). Thus, when executing the corresponding solver statement,

```
solve model maximizing j using nlp;
```

GAMS expands and stacks the system of equations and it solves it as a by-product of a pseudo-optimization.