Draft Version of GAMS Short Tutorial within "Applied Public Economics"

Please send comments to:
boehringer@zew.de

An Illustrative Introduction to GAMS

by

Christoph Böhringer and Wolfgang Wiegard

Preface

The final chapter provides an illustrative introduction to GAMS which is the programming system (language) used for numerical implementation and solution of the economic models discussed in our book. This tutorial is designed to provide the reader with a quick and intuitive understanding of how algebraic model formulations can be expressed under GAMS. Rather than giving a comprehensive overview we will motivate basic GAMS language components and GAMS syntax rules along the model examples of our book.[1] In order not to overload the reader with "unnecessary" information we explain the GAMS language just as much as is needed for translating our algebraic statement of problems into (GAMS) computer-readable format. This implies that the structure of the tutorial will reflect the sequence of economic problems presented in our book: We will begin with the GAMS implementation of the 2x2x1-model and describe additional GAMS features for subsequent models only to the extent necessary. We believe that this approach accommodates a straight computational access to learning economics and does not distract people with programming details.

Before we start with our tutorial we want to recall why we consider GAMS as a very convenient modeling system for doing computational economics:

1) The GAMS syntax is very close to the standard algebraic representation of economic models and largely self-evident. Therefore the GAMS user will find it quite intuitive to translate his algebraic problem into an equivalent GAMS program.

[1] We refer to the GAMS Users's Guide (Brooke, Kendrick, Meeraus 1996) for an ordered comprehensive description of the GAMS .

2) The use of (algebraic) sets and detached-coefficient matrix notation makes GAMS a very efficient tool for representing large dimensional models using only a few lines of code.
3) Separation of logic and data allows that a problem - once correctly specified in logical terms - can be easily scaled up to higher dimensions. This provides an efficient environment for developing large-scale models from little "toy" models.
4) The problem (model) formulation under GAMS is independent of the numerical solution. GAMS passes on the problem definition to separate solver programs which return solution values back to the GAMS program. The solution process is a black-box to the model builder who is typically not interested in algorithmical details. The only thing a model builder has to care about is that the solver he selects must be consistent with the mathematical problem type underlying his problem formulation. The commercial solvers available under GAMS allow to tackle types of models belonging to different classes of mathematical programming such as (non-)linear programming, (relaxed) mixed integer programming, (non-)linear systems of equations, mixed complementarity problems, etc.

---

1 The 2x2x1-Model in GAMS Notation

Our first economic problem to be translated into GAMS is the basic 2x2x1- model described in chapter 1: On the production side of the economy two producers maximize profits using capital and labor inputs subject to technological constraints. On the demand side a representative consumer allocates his income from fixed labor and capital endowments to maximize the utility of consumption. Production possibilities are given by linearly homogeneous functions (constant returns to scale). All the agents behave according to the competitive paradigm, i.e. they take market prices as given. The equilibrium problem then is to determine a prices, production levels and income which yield a competitive equilibrium consistent with the specified technologies, preferences and resource endowments.

The algebraic specification of this problem can be separated into three basic components:

1) *Definition of Problem Dimensionality:* The dimensions of the problem are specified by sets which summarize multiple entities of the same type and denote the domain of

decision variables. Algebraically these sets are controlled by set indices. In our 2x2x1-model the relevant sets are goods (production sectors) and factors.[2]

2) *Declaration of Decision Variables and Data Items:* Decision variables to be solved for must be declared. In addition one has to declare the data items, i.e. parameters, which are later on used for parameterization. Typically, the declaration of decision variables and parameters is indexed over associated sets to provide a compact problem description. Decision variables for our 2x2x1-model include market-clearing prices of factors and goods, production levels or income. Examples of parameters are initial factor endowments or information on technologies or preferences such as efficiency parameters or substitution elasticities.
3) *Specification of Functional Relationships:* Relationships between the various decision variables are described by means of equations (inequalities). For example, equations are used in our 2x2x1-model to state production possibilities (cost functions).

The basic components of the algebraic model description are listed in Table 1 on the left-hand side. To create a first intuition of how closely the GAMS specification of the problem is related to the standard algebraic formulation, the right-hand side presents the equivalent GAMS code. Like the algebraic formulation, the GAMS specification follows self-evident rules. For example, you will first have to define the items of the model, such as sets, parameters or decision variables before you can refer to them. Note that GAMS keywords (e.g.: For reasons of transparency, entities of the problem are grouped by type. Internal documentation, such as comments[3], in-line explanation of set elements, parameters, or variables, will significantly increase the understanding of the problem for external readers.

Table 1 gives just a blue-print description of a simple 2x2x1-economy for linearly homogeneous production technologies (constant returns to scale production). This description is not sufficient to perform a numerical calculation of the competitive market equilibrium.

[2] Note that the algebraic exposition below does not employ a set index for the factors labor and capital (e.g. f ={K,L}). Instead, we refer explicitly to these two primary factors. This representation follows the standard textbook exposition of models which only include labor and capital as factors.
[3] Comments - indicated to GAMS with an * in the first row - can be positioned at any point of the program.

Table 1: Standard Algebraic Specification of the Generic 2x2x1-Model and Associated GAMS Code

Standard Algebraic Notation	GAMS Notation
Definition of Problem Dimensionality	
Indices: i = goods {1,2}	SETS i goods / 1 good_1, 2 good_2/;
Declaration of Decision Variables and Data Items	
Decision Variables:	Positive Variables:
Y_i := total output of good i in sector i	Y(I) total output of good i in sector i,
X_i := total consumption demand of good i	X(I) total consumption demand of good i,
K_i := conditional capital demand in sector i	KY(I) conditional capital demand in sector i,
L_i := conditional labor demand in sector i	LY(I) conditional labor demand in sector i,
K_i := total capital demand in sector i	K(I) total labor demand in sector i,
L_i := total labor demand in sector i	L(I) total capital demand in sector i,
M := total income	M total income,
p_i := price of good i	P (I) price of good i,
w := wage rate	w wage rate,
r := capital rental rate	r capital rental rate;
where $Y_i , X_i , K_i , L_i , M, p_i, w , r \geq 0$	
Exogenous Data:	PARAMETERS
L := labor endowment	LBAR labor endowment,
K := capital endowment	KBAR capital endowment;

Specification of Functional Relationships

Commodity Market Equilibrium

$Y_i = X_i(p_1, p_2, M) \quad i=1,2$ | Y(I) =E= X(I)

Income Definition

$M = r\underline{K} + w\underline{L}$ | M =E= r * KBAR + w * LBAR

Capital Market Equilibrium

$K_i^Y(r,w)\, Y_i = \underline{K} \quad i=1,2$ | SUM(I, KY(I) * Y(I)) = E= KBAR

Zero-Profit Condition

$p_i = r\, K_i^Y(r,w) + w\, L_i^Y(r,w) \quad i=1,2$ | P(I) =E= r * KY(I) + w * LY(I)

Numeraire

$w = 1$ | w =E= 1

Capital Demand

$K_i = K_i^Y(r,w)\, Y_i \quad i=1,2$ | K(I) = KY(I) * Y(I)

Labor Demand

$L_i = L_i^Y(r,w)\, Y \quad i=1,2$ | L(I) = LY(I) * Y(I)

We need a concrete choice of underlying technology types and preferences as well as data assignment to exogenous parameters such as efficiency parameters or factor endowments. After having decided on the functional forms and model parameterization, we are ready to compute the endogenous (decision) variables which characterize a market equilibrium for our economy. By using GAMS we do not have to determine the solution on our own[4] but let GAMS take over the computational work. With respect to the problem solution, the only task we have to care about is to make a formal model declaration and to issue a solve command . The model declaration tells GAMS which algebraic equations state the model. Within the solve command we specify the type of algebraic problem we look at, e.g. a system of nonlinear equations. Finally, we come to the core of computational economics, i.e. looking carefully at solution values and give them an economic interpretation. To this purpose we must be able to read the standard solution output by GAMS and to generate instructive "tailored" solution reports.

Table 2 provides a schematic overview of the various components involved in the computational approach to our 2x2x1-model under GAMS. Note that the „generic“ set-up of Table 1 is completed by additional components such as data assignment, formal model declaration or solution (GAMS output) handling.

Table 2: Basic Components of a GAMS model

GAMS components:	**Described in :**
Sets Declaration Assignment of set members	Section 1.1.1
Parameters (Data) Declaration Assignment of values	Section 1.1.2
Variables Declaration Assignment of type	Section 1.1.3
Equations Declaration Definition	 Section 1.2.1 Section 1.2.2

[4] We could try to solve the model analytically or come up with some iterative numerical solution approach if analytical tractability is not given.

GAMS components:	Described in :
Running a Model	
Model definition	Section 1.3.1
Assignment of bounds / initial values to variables	Section 1.3.2
Display statement	Section 1.3.3
Solve statement	Section 1.3.4
Invoking a GAMS run	Section 1.3.5
GAMS Output and Solution Report	
Compilation output and compilation errors	Section 1.4.1
Execution output and execution errors	Section 1.4.2
Solve output and solve errors	Section 1.4.3

Table 3 represents the full GAMS code for our 2x2x1-model assuming Cobb-Douglas technologies and preferences.

Table 3: Comprehensive GAMS Code for the 2x2x1-Model with a Cobb-Douglas Specification of Technologies and Preferences

```
$TITLE: 2x2x1_MODEL

*       Set declaration and assignment
SETS
        I               goods /1 good_1, 2 good_2/;

*       Data declaration and assignment
PARAMETERS
        A(I)            efficiency parameter /1 1.96, 2 1.889 /,
        ALPHA(I)        capital value share  /1 0.625, 2 0.667/,

        BETA(I)         expenditure share;

SCALARS
        KBAR            capital endowment    /50/,
        LBAR            labor endowment      /30/;

*       Assign value shares to expenditure
BETA("1") = 0.625;
BETA("2") = 1 - 0.625;

*       Declaration of endogenous variables
POSITIVE VARIABLES
        Y(I)    total output of good i in sector i,
        X(I)    total consumption demand of good i,
        KY(I)   conditional capital demand in sector i,
        LY(I)   conditional labor demand in sector i,
        K(I)    total capital demand in sector i,
        M       total income,
        P(I)    price of good i
        r       capital rental rate
        w       wage rate;


FREE VARIABLES
        DUMMY   dummy objective variable;
```

```
*       Declaration and specification of equations
EQUATIONS
        EQU1(I)      commodity market equilibrium,
        EQU2         income definition,
        EQU3         capital market equilibrium,
        EQU4(I)      zero profit condition,
        EQU5(I)      conditional capital demand,
        EQU6(I)      conditional labor demand,
        EQU7(I)      total capital demand by sector,
        EQU8(I)      consumption demand,
        EQU9         dummy objective function;


EQU1(I)..       Y(I)            =E= X(I);

EQU2..          M               =E= r*KBAR + w*LBAR;

EQU3..          SUM(I,K(I))     =E= KBAR;

EQU4(I)..       P(I)            =E= r*KY(I) + w*LY(I);

EQU5(I)..       KY(I)           =E=
      ALPHA(I)*1/A(I)*(r/ALPHA(I))**ALPHA(I)*(w/(1-ALPHA(I)))**(1-ALPHA(I))/r;

EQU6(I)..       LY(I)           =E=
  (1-ALPHA(I))*1/A(I)*(r/ALPHA(I))**ALPHA(I)*(w/(1-ALPHA(I)))**(1-ALPHA(I))/w;

EQU7(I)..       K(I)            =E= KY(I)*Y(I);

EQU8(I)..       X(I)            =E= BETA(I)*M/P(I);

EQU9..          DUMMY           =E= 0;



*       Model declaration
MODEL MOD_2x2x1 /ALL/;

*       Choice of numeraire
w.FX     =1;


*       Assignment of lower bounds on variables
P.LO(I)   = 0.001;
r.LO      = 0.001;

*       Initialize level values for factor prices to ease solution
r.L       = 0.5;


*       Solve statement
SOLVE MOD_2x2x1 USING NLP MAXIMIZING DUMMY;

*       Report solution values
Display K.L, X.L, Y.L, P.L, r.L, w.L;

*       Specify solution report
PARAMETER
        OUTPUT  summary solution report for goods;

OUTPUT("SUPPLY",I) = Y.L(I);
OUTPUT("DEMAND",I) = X.L(I);
OUTPUT("PRICES",I) = P.L(I);

DISPLAY OUTPUT;
```

For the detailed discussion of the GAMS components and syntax below it is useful to keep some general rules for GAMS programming in mind:

1) The creation of GAMS entities (components) involves always two steps: a declaration and an assignment. The declaration involves indicating the existence of the entity type and giving it a name. The naming convention for entities is that the name must start with a letter which then can be followed by up to nine more letters or digits. The assignment means that a specific value (e.g. data assignment) or form (e.g. function specification) is attributed to the declared entity. Models can be developed and documented simultaneously because GAMS allows the user to include explanatory text as part of the declaration of any entity. It is useful to choose documentary text that is self-evident. The documentary text is optional but highly recommended for transparency.[5]
2) A GAMS entity may not be declared more than once. This is a GAMS error check to prevent that the same name is used for two different things.
3) As to typography GAMS is quite flexible. Users are e.g. free to place blank spaces and lines upon their needs. GAMS allows multiple lines per statement and vice versa. GAMS is not case sensitive.
4) Semicolons should always be used to indicate the end of a GAMS statement.[6]

2. Specifying Key Entities: Sets, Parameters, Decision Variables

2.1 Definition of the Model Dimensions: The SET Statement

The dimensionality of a model written in GAMS can be most easily captured by set elements which directly correspond to the indices in algebraic notation. Specifying algebraic indices in GAMS involves the SET statement followed by the names of sets (indices) and the assignment of set members, i. e. the range or domain of the indices.

Our basic 2x2x1-model contains a set of goods (or likewise production sectors) as a controlling index of the algebraic model formulation. The associated GAMS specification is:

```
Sets   i  goods      / 1 good 1, 2 good 2 /;
```

After indicating the creation of algebraic indices by the keyword SET (or SETS), each set definition is composed of two components (declaration and assignment):

[5] When compiling the model, the GAMS compiler does not interpret the text, but it saves the text and 'parrots' it back to you whenever associated values are displayed.

[6] The semicolon after a GAMS statement may be omitted if it is followed directly by a GAMS keyword (see GAMS Users's Guide for the full list of keywords) but it is good programming style to always employ the semicolon for terminating a statement.

1) the name of the set and an optional description of what the set stands for, and
2) a list of set members enclosed between a pair of slashes[7]. Multiple set elements are separated by commas where optional internal documentary text may accompany each set element).The set elements are stored as character strings, so the elements of set *i* above are not numbers but strings "1" and "2".

Note that the dimensionality of our model is in formal terms given by 2x2x1, that is by the Cartesian product of the number of goods, factors and households in the sample economy. We can capture the different dimensions through indices in the algebraic formulation or likewise sets in the GAMS notation. The current example then would involve the definition of three sets. However, in compliance to the standard textbook exposition of the basic 2x2x1-model, we only use set algebra for a compact representation of goods (production sectors) and will refer to factors (labor and capital) explicitly.[8]

It is obvious that the algebraic-type use of sets, makes GAMS very convenient for large-scale modeling. After having develop the structure of a problem at lower dimensionality, it is a simple matter to extend the model's dimensionality by just adding further set elements. For large-scale models it could become tedious to define all the set elements explicitly. Therefore GAMS offers the possibility for a compact definition of elements which follow a sequence. Imagine, for example, that our model would encompass the 58 production sectors of the standard German input-output tables. Then a compact set statement using the * operator in GAMS is:

```
Set    i  goods      /1*58/ ;
```

2. 2 Declaration of Data Entities with Data Assignment: The PARAMETER Statement

Data (numbers) that will be used in a model for parameterization are primarily associated with the GAMS data types PARAMETER.[9] Data can be either directly entered by specifying numbers as part of the data type definition. Another way of incorporating data are assignment statements "outside" the parameter declaration which allow to define or alter values associated

[7] Algebraically the set statement reads as $i = \{1,2\}$, $j = \{L, K\}$. Note that GAMS uses slashes '/' rather than curly braces '{}' to delineate the set. The simple reason for this is that not all computer keyboards have keys for curly braces.

[8] As a useful exercise you may do a fully set-based GAMS translation of the 2x2x1-model with a „stringent“ use of sets for capturing the model's dimensionality.

[9] Special subtypes of PARAMETERS are SCALARS which refer to unindexed, zero-dimensional data items and TABLES which accommodate data entry in table form.

with any parameters. It is self-evident that the "outside"-assignment method only can be applied to parameters which have been declared beforehand.

The keyword PARAMETER indicates that data items will be defined. The PARAMETER statement is associated with indexed (subscript) data items. To enter indexed data within the PARAMETER statement one has first to declare the name of the data item including its indices (domains).[10] The second optional step is to insert documentary text for the parameter. The third step involves the data assignment within slashes: Commas must separate element-value pairs from each other within the same line. It is possible to skip the commas when element-value pairs are entered in separate lines.

For a parameter with zero-dimensionality one has to use the GAMS statement SCALAR which explicitly indicates that unindexed data items will be defined. After entering the keyword SCALAR we have to declare the names of the scalar, then include a verbal description (optionally) and finally we can assign a numerical value enclosed in slashes.
For the 2x2x1-model in Table 3 the declaration of parameters (incl. scalars) and the associated data assignments are as follows:

```
*       Data declaration and assignment
PARAMETERS
        A(I)            efficiency parameter /1 1.96, 2 1.889 /,
        ALPHA(I)        capital value share  /1 0.625, 2 0.667/,
        BETA(I)         expenditure share;

SCALARS
        KBAR            capital endowment    /50/,
        LBAR            labor endowment      /30/;

*       Assign value shares to expenditure
BETA("1") = 0.625;
BETA("2") = 1 - 0.625;
```

The example above illustrates two possibilities for data assignment. Firstly, we can combine data item declaration and data within an extended data declaration block (see e.g. the declaration and assignment of values for efficiency parameters ALPHA(I)). The order of element-value pairs in indexed data assignments is free and need not correspond to the

[10] This invokes a very useful GAMS feature, the so-called domain-checking. Whenever an entity is specified over a previously defined set, the GAMS compiler references over sets or specific elements of this entity are consistent with the element is in the domain of the superior set. This turns out to be very useful for checking internal consistency of a problem.

ordering of the respective set elements in the previous SET statement.[11] Note that the default value for all parameters is zero such that only entries distinct from zero must be included in the element-value list. Secondly, we can assign values by direct data assignments as is the case for the parameter *BETA(I)*. Note that specific elements of domains are referred to by just enclosing the element names in quotes:

The domain of the parameters can be spread to multiple dimensions For example, if we were to extend our closed economy model to multiple regions and assign country- and sector-specific data for efficiency parameters, the following statements would do the job:

```
Sets
       r      regions / North, South/ ;
Parameters
       R_ALPHA (i,r) sector-specific capital in various regions
              /      1.NORTH       1.96
                     1.SOUTH       1.5
                     2.NORTH       1.889
                     2.SOUTH       1.45
                      /;
```

The domain of *R_ALPHA* here is the set of ordered pairs in the Cartesian product of `i` and `r`.

2.3 Declaration of Decision Variables

Decision variables of a model written in GAMS are declared using the VARIABLE(S) statement After naming the variable and defining its domain, we can optionally add a documentary text. GAMS generates each instance of the variable in the domain. In order to increase the transparency of a complex model it is recommended to choose "speaking" names and not omit documentary text. The sign condition (type) of a variable can be simultaneously defined by using corresponding prefixes in the VARIABLES statement as listed in Table 4. [12]

Table 4: Types of Variables with permissible Range

GAMS Variable Type Statement	**Allowed Range of Variable (default lower and upper bounds)**
Free Variable(s)	$-\infty$ to $+\infty$
Positive Variable(s)	0 to $+\infty$
Negative Variables(s)	$-\infty$ to 0
Binary Variables(s)	0 or 1
Integer Variable(s)	0,1,...., n (with 100 as the default upper bound)

In our 2x2x1-model no negative values are allowed for the decision variables and therefore we declare them as POSITIVE VARIABLES:

[11] For the sake of transparency it might however be useful to keep the initial ordering.

[12] It is also possible to first declare all decision variables using the VARIABLES statement and then specify the sign condition later on through additional statement such as FREE VARIABLES, POSITIVE VARIABLES….

```
POSITIVE VARIABLES
        Y(I)    total output of good i in sector i,
        X(I)    total consumption demand of good i,
        KY(I)   conditional capital demand in sector i,
        LY(I)   conditional labor demand in sector i,
        K(I)    total capital demand in sector i,
        M       total income,
        P(I)    price of good i
        r       capital rental rate
        w       wage rate;
```

There is one subtlety under GAMS owing to the fact that the GAMS language was originally developed for solving optimization problems. When we use solution algorithms for mathematical programming problems we must specify our problem in formal terms as a mathematical program. If we state the equilibrium conditions as a system of equations - which is the case for our 2x2x1-model - we have to declare an explicit objective function even though there is no argument to be maximized or minimized.[13] GAMS then reads the equilibrium conditions as side-constraints which must be satisfied when optimizing a dummy objective function. By definition, the variable associated with an objective function must be of the FREE type which explains the additional FREE VARIABLE(S) statement in the full GAMS program for the 2x2x1-model:

```
FREE VARIABLES
        DUMMY   dummy objective variable;
```

3. Specifications of Functional Relationships: Equations

The use of sets provides a very efficient way of creating functional relationships under GAMS. Equations that share the same algebraic structure can be created with just a single GAMS statement.[14] After indicating the domain of an equation type by set indices GAMS

[13] It should be noted that treating equilibrium conditions as side constraints and solving them as part of a dummy (non-)linear optimization problem is typically not an efficient and robust strategy for large-scale problems. The "fake" specification of programming problems where the feasible space is restricted to a single vector (i.e. the solution) requires lots of sophisticated bound setting and initializing of decision variables to prevent the solver from having problems. Meanwhile GAMS features the mixed complementarity format (MCP) which provides a natural way for expressing a variety of economic models for both markets and games. Commercial algorithms for solving MCPs have proven to be quite reliable and efficient.

[14] Note that the keyword `EQUATION` under GAMS is used for denoting both equality and inequality relationships.

automatically generates each equation. Equations are declared and defined in separate statements.

3.1 Declaration of Equations

Equations are declared using the EQUATION(S) statement. First the keyword EQUATION(S) indicates the creation of equations. Then each equation (type) must be given a name, followed by the domain and optional documentary text. In order to specify the equilibrium conditions for our 2x2x1-model we first have to declare the following equations under GAMS:

```
EQUATIONS
        EQU1(I)       commodity market equilibrium,
        EQU2          income definition,
        EQU3          capital market equilibrium,
        EQU4(I)       zero profit condition,
        EQU5(I)       conditional capital demand,
        EQU6(I)       conditional labor demand,
        EQU7(I)       total capital demand by sector,
        EQU8(I)       consumption demand,
        EQU9          dummy objective function;
```

3.2 Definition of Equations

The definition of equations involves two components:

1) The name of the equation including its domain followed by the symbol '..'
2) The algebraic relationship which consists of a left-hand-side expression, the indication of the form of the equation by means of a relational operator, i.e.:

   =L= less than or equal to,

   =G= greater than or equal to, or

   =E= equal to[15]

   and the right-hand-side expression.

GAMS provides an extensive set of operators and standard functions which enables you to define any algebraic relationship. One example for an indexed algebraic operation employed in our example is the summation statement SUM. This statement is used to calculate totals over the domain of selected sets. The SUM statement involves the specification of indices and the specification of the (indexed) argument for summation. The argument can be any mathematical expression including other summations.

[15] Note that the "=E=" operator is different from "=". Whereas "= " goes along with value assignments associated to parameters or sets (before the solver is called), the "=E=" operator is associated to variables within EQUATIONS (which are evaluated within the solution process).

Assuming Cobb-Douglas technologies and preferences the equilibrium conditions for our 2x2x1-model can be stated as:

```
EQU1(I)..        Y(I)             =E= X(I);

EQU2..           M                =E= r*KBAR + w*LBAR;

EQU3..           SUM(I,K(I))      =E= KBAR;

EQU4(I)..        P(I)             =E= r*KY(I) + w*LY(I);

EQU5(I)..        KY(I)            =E=
      ALPHA(I)*1/A(I)*(r/ALPHA(I))**ALPHA(I)*(w/(1-ALPHA(I)))**(1-ALPHA(I))/r;

EQU6(I)..        LY(I)            =E=
  (1-ALPHA(I))*1/A(I)*(r/ALPHA(I))**ALPHA(I)*(w/(1-ALPHA(I)))**(1-ALPHA(I))/w;

EQU7(I)..        K(I)             =E= KY(I)*Y(I);

EQU8(I)..        X(I)             =E= BETA(I)*M/P(I);

EQU9..           DUMMY            =E= 0;
```

Note that *EQU9* which defines a dummy objective function has no real meaning in our model specification. It just must be included for the formal reasons mentioned in section 1.1.3 when we solve the equilibrium conditions as the side constraint of a NLP problem. The objective function - in our case the scalar value of 0 - is associated with a free scalar variable to be optimized in the solve statement of the model (see section 1.3.3 below).

4 Running a Model: The Final Steps

After having specified sets, data items, variables and equations, there are some additional steps before we can run the model and interpret its solution.

4.1 Formal Model Definition: The MODEL Statement

We formally create the model by telling GAMS some name of the model and the equations it consists of. The associated GAMS command is the statement MODEL followed by the unique model name and the set of equations in slashes which form the model. In our example, we use the keyword /ALL/ which effects that all equations being defined previously are included in the model:[16]

[16] The option to select specific equations within slashes allows for the formulation of different models within a single GAMS input file. This can be very useful when working with submodels from an overall comprehensive list of functional relationships.

```
MODEL MOD_2x2x1 /ALL/;
```

4.2 Initializing Variables and Setting Bounds

In many applications it is necessary to set specific bounds on variables in order to rule out undefined operations such as division by zero. Another purpose of specifying bounds is to narrow the solution space in order to avoid that the solution algorithm get stuck into infeasibilities. For the latter reason, it may be also necessary to assign an initial solution from which the solver can start searching for the optimum. There are four suffixes associated with each variable which allow to set user-defined lower and upper bounds as well as the initial level values for variables.

Table 5: Selected GAMS Suffixes for VARIABLES

GAMS Variable Suffix		**Default Values**
.LO	specification of lower bound	Determined by variable type
.UP	specification of upper bound	Determined by variable type
.FX	specification of lower and upper bound	Fixed to specified value
.L	initialization of level values	0 if 0 is in the bounded range, bound closest to zero if 0 is not in the bounded range

Note that the *.LO* and *.UP* fields are fully controlled by the GAMS user whereas the *.L* field can be initialized but then is endogenous to the solver. The bound assignment with the *.FX* suffix effects that both the lower and upper bounds are set to be equal a user-defined value.

In our 2x2x1-model we set positive lower bounds on the positive price variables in order to avoid division by zero. In addition, we initialize the starting values of for factor prices different from zero (zero is the default) to prevent solution problems. Note that we use the *.FX* suffix to select the numeraire for our 2x2x1-model[17]:

```
*       Choice of numeraire
w.FX     =1;


*       Assignment of lower bounds on variables
P.LO(I)    = 0.001;
r.LO       = 0.001;

*       Initialize level values for factor prices to ease solution
r.L        = 0.5;
```

4.3 The SOLVE Statement

After we have defined the model and bounded as well as initialized decision variables, we can call the solver using the SOLVE statement:

```
SOLVE MOD_2x2x1 USING NLP MAXIMIZING DUMMY;
```

The SOLVE statement is composed of four element:

1) It tells GAMS which model to solve (in our case: *MOD_2x2x1*).
2) It selects the solver to use (in our case: *NLP* solver).
3) It indicates the direction of the optimization (either *MINIMIZING* or *MAXIMIZING* - in the case of the dummy objective function we could use both) ,
4) It refers to the objective variable (here: *DUMMY*).

The SOLVE statement in GAMS accommodates a convenient separation between the model formulation and the solution process which is a black-box to the model builder. We only have to take care that the solver selected must be consistent with the mathematical problem type underlying our specification.

4.4 Writing Data: The DISPLAY Statement

The (optional) DISPLAY statement allows to write data into the listing file which can be very useful for data checks and user-designed solution reports. We can use the DISPLAY statement in connection with sets, parameters, variables, and quoted text strings at any location of our program. The output generated for sets is a list of set elements. The output produced for variables and data items consists of labels and associated values. In our 2x2x1-model (see Table 3) we have specified two DISPLAY statements after the SOLVE statement because we want to investigate the values of decision variables after having been returned by the solver. The first DISPLAY statement invokes a printout of the level values of the decision variable.[18] The second DISPLAY statement effects that a user-specified 2-dimensional parameter which summarizes some computational results gets printed. GAMS will automatically format this printout in a two-dimensional table with appropriate headings:

```
*       Report solution values
Display K.L, X.L, Y.L, P.L, r.L, w.L;
```

[17] Instead of defining the numeraire by means of equal upper and lower bounds we could have incorporated an additional equation in our GAMS program where we postulate that the numeraire has to be equal to some given value (typically unity). The former approach is however much more efficient.
[18] Note that we have to communicate our interest in the level value to GAMS by using the .l suffix.

```
*       Specify solution report
PARAMETER
        OUTPUT  summary solution report for goods;

OUTPUT("SUPPLY",I) = Y.L(I);
OUTPUT("DEMAND",I) = X.L(I);
OUTPUT("PRICES",I) = P.L(I);

DISPLAY OUTPUT;
```

Note that only the non-default values for variables and data items are displayed. The labels of set elements for DISPLAY arguments are always ordered according to the first appearance of labels within the GAMS program which can - particularly in large programs - lead to a undesired label order in displays. The easiest way to assure a specific order of labels is to declare a set which serves the sole purpose to list all the labels in the order that is needed and to make this set the very first declaration in the GAMS program.[19]

4.5 Invoking a GAMS Run From the System Prompt

To run the model, i.e. to trigger compilation and execution by GAMS and its solver subsystem, we must enter the command GAMS followed by the file name which includes our GAMS program from the system prompt.

5 GAMS Output and Solution Report

By default, the output is written to a listing file having the same name as the original GAMS program file but employing the extension *.lst* (on DOS machines).[20] The GAMS output reported in the listing file includes three major components: the compilation output, the execution output and the output produced by the solve statement. These output components correspond to the three phases invoked by a GAMS run. During the compilation phase the GAMS program will be checked for syntax and consistency. In the second phase of the GAMS processing, GAMS is executing data transformations and model generation. During the third phase, the solve statement effects that the model is translated to a format appropriate

[19] With respect to DISPLAY arguments which have several dimensions GAMS allows to modify the number of row and column labels in the listing file as well as the accuracy of numerical values to be displayed (see GAMS user's guide).

[20] There are several options available at the command line to control: (1) the specific GAMS run, (2) system settings, (3) input file processing, (4) output in listing file, or (5)other files. For a comprehensive list of command line options including feasible option values see the GAMS user's guide.

for the selected solver, then passes on control to the solver subsystem and waits while the problem is solved and solution values are returned. The above sequence of phases will be interrupted at any stage whenever an error occurs. The type of errors can be assigned to three categories associated with the phase of the overall GAMS run: compilation errors, execution errors and solve errors. The location and type of error will be pinpointed in the respective output component. Errors are indicated in the output report by four asterisks ****** at the beginning of a line in the output listing. For compilation and execution errors a *$*-sign indicates the potential source of error directly below the offending GAMS expression followed by a numerical error code which is explained in length at the end of the listing file (Appendix A includes a list of all error messages specified in the version 2.25 of GAMS). The GAMS novice should keep in mind two general rules for debugging compilation and execution errors:

1) The first error encountered by GAMS during compilation can create lots of consequential error messages. Therefore one should always fix the first error and do not care for the others.
2) Most of the error messages provide instructive information for correcting the errors. However, the GAMS error list can not cover all potential sources for errors. Therefore we may have to search for the nature of the error on our own whenever the GAMS error message is obviously inappropriate.

5.1 Compilation Output and Compilation Errors

By default the first part of the compilation output is an echo print, i.e. a copy of the original GAMS program supplemented with line numbers. Whenever GAMS encounters a compilation error, the error message is displayed after the echo print section. Most compilation errors are caused by simple mistakes such as forgetting to declare an identifier, putting indices in the wrong order, omitting a necessary semicolon, misspelling a label a.s.o. The compilation output includes additional information, so-called maps, on the structure of the model. A first map lists all the identifiers of the program in alphabetical order, associates them with their item type, shows the line numbers where the symbols appear, and finally classifies each appearance. The second map is again a listing of the identifiers (symbols), but now grouped alphabetically by type and supplemented with their explanatory texts.

5.2 Execution Output and Execution Errors

Except for execution errors, the report produced during the execution phase only includes output associated with the DISPLAY statement. Execution errors stem from illegal data manipulations such as division by zero or taking the log of a negative number.

5.3 Solve Output and Solve Errors

The solution phase produces an extensive output whose single parts can be summarized as follows:

1) Equation Listing: The equation listing shows the individual constraints that have been generated from the equation blocks or the GAMS program file. The individual constraints are listed with their current values for set elements and parameters after having performed all the data manipulations.[21] This listing allows to check precisely if the model stated by the equations correspond to the problem which one wanted to specify. Note that for nonlinear equations, the GAMS equation listing reports first-order Taylor approximations at the starting values of the variables.

2) Column Listing: The column listing provides a summary of the coefficients associated with each variable.[22]

3) Model Statistics: This part of the solve output provides statistics such as the size of the model (i.e. number of equations and variables), the degree of its non-linearity or the time requirements for generating and executing the model.

4) Solve Summary: The solve summary contains important information on the solution process itself (SOLVER STATUS message) and the characteristics of the problem solution (MODEL STATUS message). The SOLVER STATUS tells whether the solver terminated in a normal way or has been interrupted due to limits (e.g. time limit, iteration limits)[23] or internal difficulties. The MODEL STATUS gives information on the type of solution (e.g.: optimal, locally optimal) or the type of problem associated with a solution failure (e.g. infeasible).

5) Solution Listing: The solution listing reports the results for the decision variables which constitute a solution to the specified problem whenever the SOLVER STATUS and

[21] Note that, by default, only three specific instances of a generic equation are reported. If we want to see *r* instances we must add the GAMS statement *option limrow = r;* within the GAMS program.

[22] The default is to show only the first three entries for each variable such that one has to add the GAMS statement *option limcol = r;* when we to have *r* entries being displayed.

MODEL STATUS confirm the existence of a solution. The actual solution value of variables are reported as level values. In addition, the solution listing provides information on the (default or user-specified) lower and upper bounds as well as the computed marginals[24] for each variable. The solution listing for equations has the same structure as for variables. Each equation is reported with values for the lower bound, the level value, the upper bound value and the marginal value. Note, that the lower and upper bound for equations is obtained from the constant right-hand side value and from the relation type of the equation.

Solve errors may have two sources. Either they are due to problems transforming the GAMS model in a solver compatible format or they stem from problems faced by the solver when evaluating nonlinear functions or computing derivatives.

6 SELECTED „ADVANCED“ FEATURES UNDER GAMS

6.1 The ALIAS Statement

An important feature for many operations is the ALIAS statement, which gives another name to a previously declared set. In our example the statement

```
Alias (f,g) ;
```

effects that *f* and *g* can be equally used as the index set for sectors. The ALIAS statement allows for interactions between elements of the same set which is useful for many problems with higher dimensionality.[25]

6.2 The Dollar Operator

In many model specifications it is essential to define exceptions in indexed assignment statements or in indexed equation definitions. For example, users may want to define equations or assign values to parameters only for such domain elements which satisfy some logical or numerical conditions. In GAMS the operator *$* (so-called dollar operator) offers a

[23] The user can increase the time or iteration limits through specific option statements (see GAMS user’s guide).
[24] The marginal (dual) value of variables or equations contains information about the rate at which the objective value will change of it if the associated bound or right hand side is changed.
[25] For example, the description of the production side of an economy often includes various sectors where intermediate supplies from sectors *i* to sectors *j* must be accounted for.

comprehensive capability for exception handling. Following we provide some illustrative examples for the use of the dollar operator, more detailed information can be found in the GAMS user's guide.[26]

```
C0(I)$(B0(I) NE 0) = A0(I)/B0(I);
```

In the example above we avoid division by zero whenever an element of the parameter *B(I)* is zero. Otherwise GAMS will prompt an error message during compilation. To understand the use of the dollar operator one has to keep in mind two basic features of the GAMS language:
The result of a logical operation is zero if the assertion is false and one if it is true.
GAMS lacks an explicit Boolean data type. When interpreting the result of a logical condition GAMS takes zero as false and non-zero as true.

Note that the following statement is then equivalent to our first statement because the expression associated with the dollar operator is interpreted as true for non-zeros of the parameter *B0(I)* and as false for zero values:

```
C0(I)$B(I) = A0(I)/B0(I);
```

The dollar operator can be used to do implicit if-else statements.[27] Assume for example that we have defined a scalar *Flag* which can be assigned a value of zero or a non-zero-value at some point of the program. The following statement then effects that *C0(I)* takes over the value of *A0(I)* whenever the *Flag* is non-zero and assigns a value of *B0(I)* otherwise:

```
C0 (I) = A0(I)$(Flag)+ B0(I))$(NOT Flag);
```

Often the controlling item of a dollar controlled index operations is a set. For illustration, consider a case where we sum up the supply values of single goods in order to obtain the supply values of a composite good subject to a predefined mapping:

```
SETS  I     single goods    / ELE, GAS, OIL, AUTO, AIR, SHIP, BEER, MILK/
      J     composite goods / ENE, TRAF, BEV/;

      MAP(I,J) mapping between single goods and composite goods
                        /(ELE,GAS,OIL).ENE,
                         (AUTO,AIR,SHIP).TRAF,
```

[26] For an intuitive interpretation the dollar operator may be read as "if" or "such as".
[27] Alternatively GAMS features an explicit IF-ELSE statement (see GAMS user's guide) which might make the GAMS code more readable in complex cases.

```
                          (SHIP,BEER,MILK).BEV/;

PARAMETERS
      SUPPLY (I)         monetary value of supply,
      AGG_SUPPLY(I)      aggregate supply;
SUPPLY(I) = 1;

*   Aggregate supply values according to mapping
     SUPPLY(J)     = SUM(I$MAP(I,J), QUANTITY(I));
```

Finally we provide an example where the dollar operator is used to control the domain of definition of an equation:

```
EQU(I)$(SIGMA(I) NE 1).. Q(I)=E= (1/A(I))* ( alpha(I)*r**(1-sigma(I))+
                         (1-alpha(I))*w**(1-sigma(I)) )**(1/(1-sigma(I)));
```

In this case the constraint is only produced for those elements of the set *I* for which *sigma(I)* is not equal to unity.

6.3 Error Checks: ABORT and $EXIT Statements

The ABORT statement can be used in combination with the dollar operator to terminate the GAMS program if some logical or numerical conditions does not hold. This is extremely helpful for checks of data integrity. In the context of this book it should be noted that equilibrium analysis often relies on a social accounting matrix as reference (benchmark) equilibrium which also serves to calibrate parameters of functional forms. It is straightforward then that a consistent equilibrium model specification should replicate the benchmark equilibrium when just plugging benchmark prices and quantities into the algebraic form. However this benchmark replication check will be only meaningful when data consistency has been assured beforehand. Otherwise one can face a deviation from equilibrium conditions which is not due to a false algebraic specification but due to bad data. Therefore it is highly recommended to specify data checks whenever data which is subject to equilibrium conditions is read in or is manipulated during the program. Following we provide an example where we check the data consistency of a social accounting matrix with respect to the identity of associated row and column sums:[28]

```
SETS
      R      rows of SAM     /R1*R5/,
      C      columns of SAM  /C1*C5/;

ALIAS (R,RR), (C,CC);

TABLE SAM(R,C)  Social Accounting Matrix
      C1    C2    C3    C4    C5
R1     1     5     3     4     2
R2     2     4     1     5     3
```

[28] In the example we use standard GAMS operators and functions which we have not yet introduced. The ORD operator returns the relative position of a member in a set and is used on one-dimensional, static, ordered sets. The SMAX operator returns the maximum value over controlling indices. The ABS function returns the absolute value of its argument.

```
R3      3      1      4      2      5
R4      4      2      5      3      1
R5      5      3      2      1      4 ;

*       Consistency check of Social accounting matrix

PARAMETER CHECK            Data check parameter;

PARAMETER CHECK(R,C);
CHECK(R,C)$(ORD(R) EQ ORD(C)) = SUM(CC, SAM(R,CC)) - SUM(RR,SAM(RR, C));

DISPLAY CHECK;

ABORT$(SMAX( (R,C), ABS(CHECK(R,C))) GE 1.E-6)"SAM DATA IS NOT CONSISTENT:",CHECK;
```

In this example we first assign the difference of associated row and column sums to the parameter CHECK. The ABORT statement will automatically prompt us with a warning and abort compilation when a row sum and its respective column sum deviate from each other.

The $EXIT statement effects an exit from compilation. This statement is often used as a "quick and dirty" way to trace back compilation errors. In the example from above, we could use the following statement to interrupt compilation and to perform an explicit data check via the DISPLAY statement in the listing file :

```
DISPLAY "This is a check for data consistency:", CHECK;
$EXIT
```

6.4 Inclusions of External Files: $INCLUDE, $BATINCLUDE and $LIBINCLUDE

The $INCLUDE option allows to insert the contents of an exogenous file at the location where the $INCLUDE command followed by the name of the file to be included appears within the main GAMS program.[29] This option can be quite useful for loading large data sets or sub-routines which the modeler wants to keep separate from its main program, e.g. for reasons of overall transparency. Compilation errors within $INCLUDE files will be pinpointed with the local line number in the respective file. As compared to the $INCLUDE statement the $BATINCLUDE and $LIBINCLUDE options allow to pass on arguments which can be used inside an include file. The syntax then is $BATINCLUDE ($LIBINCLUDE) followed by the name of the file to be included and a list of the arguments which should be passed on. The arguments are treated as character strings that are substituted by number inside the include file. A parameter substitution is indicated by using the character *%* followed directly by an integer value which corresponds to the order of parameters on the list (i.e. *%1* refers to the first argument). The only difference between $BATINCLUDE and $LIBINCLUDE is that $LIBINCLUDE uses the library include directory if an incomplete path

is given whereas $BATINCLUDE requires the full path or otherwise retrieves within the current directory.[30] In many advanced GAMS applications the $LIBCINLUDE command is used to call powerful shareware GAMS batch programs for report writing, spreadsheet interaction and plotting graphical images (see section 6.7).

6.5 The LOOP Statement

The LOOP statement allows to execute the statements within the scope of the loop for each member of the controlling set(s).[31] Loops are used whenever parallel operations are not possible, but iterative calculations are necessary.[32] For illustration, consider that in our 2x2 economy we want to apply a sequence of different ad-valorem taxes on the consumption of good *1*. After modifying the relevant equation *EQU6(I)* in the base program:

```
PARAMETERS    TAX(I) Ad-valorem tax;

EQU6(I)..       X(I)               =E= [ BETA(I)*M/(PC(I)*(1+ TAX(I)) ];
```

the iterative GAMS procedure could then be specified as follows:

```
SETS   SC     Scenario      /T05,  T10, T50, T100/

PARAMETERS TAXRATE(SC)     Counterfactual ad-valorem taxes
                           /T05  0.05,
                            T10  0.1,
                            T50  0.5,
                            T100 1/;

LOOP(SC,
       TAX("1") = TAXRATE(SC);
SOLVE MODEL_2X2 USING NLP MAXIMIZING DUMMY;
);
```

6.6 The PUT Statement

The PUT statement accommodates the output of results to different files at one time. This statement features a wide range of numeric output formats, page formatting, text and numeric justifications such that the user is quite flexible to shape the output to his requirements. Before the PUT statement can be used one must first specify the file where the GAMS output should be directed to using the FILE statement. For illustration consider our 2x2 model where we do not only want to print a summary report to the listing file (using the DISPLAY statement) but also to write a report to some additional file called *results.dat*. Be aware that as a major difference between the PUT statement and the DISPLAY statement PUT only works on

[29] Note that include files can be nested.
[30] In GAMS the default is the INCLIB directory in the GAMS system directory.
[31] Note that it is not feasible to make declarations or define equations inside a LOOP statement.
[32] A wide-spread example in applied modeling is the dynamic-recursive computation of a static model along the timepath where values such as capital stocks are updated between timesteps.

scalars and not on blocks. This means that we have to use the LOOP statement whenever we want to write blocks. In our case the GAMS code could then look like[33]:

```
OUTPUT("SUPPLY",I) = Y.L(I);
OUTPUT("DEMAND",I) = X.L(I);
OUTPUT("PRICES",I) = PC.L(I);

DISPLAY OUTPUT;

*      Specify the name of the external results file
FILE RES /RESULTS.DAT/;

*      Specify a set with the names of the items to be plotted as part of the report

SET    ITEMS /SUPPLY, DEMAND, PRICES/

*      Indicate to which file we plot now (there might be various output files in
*      use)

PUT RES;

LOOP( (ITEM,I),
              PUT ITEM.TL, OUTPUT(ITEM,I);
              PUT /;
    );
```

6.7 Environment Variables

In GAMS, an environment variables is set with the $SETGLOBAL statement followed by a variable name and a specified value. After this specification the variable can be referenced within the GAMS program by the expression *%variable name%*. Environment variables are very useful to run a GAMS model for alternative settings of key parameters from batch (sensitivity analysis). For examples we refer to the GAMS model library.

6.8 GAMS PROGRAM EXAMPLES

6.8.1 2x2x1-Model in Vector Syntax

```
$TITLE:MODEL_2X2

*       Set declaration and assignment

SETS
        I               Production sectors or goods /1  Food, 2 Others/
        F               Factors of production   / L labor,
                                                  K capital/,
        H           Set of households  /RA representative household/;


ALIAS(f,ff);


*       Data declaration and assignment

PARAMETERS
```

[33] The "/" character creates a linefeed. The identifier suffix ".TL" creates that the labels of set elements are written.

```
        A(I)            efficiency parameter  /1 1.96, 2 1.889 /,
        ALPHA(F,I)      value shares          /K.1 0.625, L.1 0.375,
                                             K.2 0.667, L.2 0.333 /,
        BETA(I,H)        expenditure shares    /1.RA 0.625, 2.RA 0.375/,
        ENDOW(F, H)      factor endowments     /L.RA 30, K.RA 50/;


*       Declaration of endogenous variables

POSITIVE VARIABLES
        Y(I)            total output of good i in sector i,
        X(I,H)          total consumption demand of good i,
        FD(F,I)         conditional factor demand in sector i,
        PF(F)           factor prices,
        M(H)            total Income,
        PC(I)           price of good i;


FREE VARIABLES
        DUMMY;


*       Declaration and specification of equations
EQUATIONS
        EQU1(I)      zero profit condition,
        EQU2(F)      market clearance for factors,
        EQU3(I)      market clearance for goods,
        EQU4(H)      income balance,
        EQU5(F,I)    conditional demand of factor f for production of good i,
        EQU6(I,H)    consumption demand for good i by household h,
        EQU7         dummy objective function;

EQU1(I)..       PC(I)                 =E= SUM(F, PF(F)*FD(F,I));
EQU2(F)..       SUM(I,FD(F,I)*Y(I)) =E= SUM(H, ENDOW(F,H));
EQU3(I)..       Y(I)                  =E= SUM(H, X(I,H));
EQU4(H)..       M(H)                  =E= SUM(F, PF(F)*ENDOW(F,H));
EQU5(F,I)..     FD(F,I)               =E=
        ALPHA(F,I)*1/A(I)*PROD(FF, (PF(FF)/ALPHA(FF,I))**ALPHA(FF,I)) /PF(F);
EQU6(I,H)..     X(I,H)                =E= [ BETA(I,H)*M(H)/PC(I) ];
EQU7..          DUMMY               =E= 0;


*       Assignment of lower bounds on variables
PC.LO(I)   = 0.001;
PF.LO(F)   = 0.001;

*       Initialize level values for factor prices to ease the solution process
PF.L(F)  = 0.5;

*       Choice of numeraire
PF.FX("L")     =1;


OPTION NLP = CONOPT;

*       Model declaration and solve statement
MODEL MODEL_2X2 /ALL/;
SOLVE MODEL_2X2 USING NLP MAXIMIZING DUMMY;


*       Report solution values
Display FD.L, X.L, Y.L, PC.L, PF.L, M.L;

*       Produce solution report

PARAMETER
        OUTPUT  summary solution report for goods;

OUTPUT("SUPPLY",I) = Y.L(I);
OUTPUT("DEMAND",I) = SUM(H, X.L(I,H));
OUTPUT("PRICES",I) = PC.L(I);

DISPLAY OUTPUT;
```

6.8.2 Usage of $SETGLOBAL in BATCH Programming

In the following we provide a simple example how we can use the $SETGLOBAL statement for external model parameterization. This can be very useful for doing sensitivity analysis without having to change anything in the core model file. In our specific example the little batch program allows to assign values to labor and capital endowment of our standard 2x2x1-economy from the DOS prompt.

→ *Batch Program For Assigning Values to KBAR and LBAR which are the SCALARS within the 2x2x1-model carrying the information on capital and labor endowment of the economy.*

```
goto start
echo  ============================================================
echo  Syntax:           run_sc kbar lbar
echo  ============================================================
echo.
echo  where "kbar" is:
echo.
echo  positive integer number (capital endowment)
echo.
echo  and "lbar" is one of:
echo.
echo  positive integer number (labor endowment)
echo.

:start
@echo off
echo  $setglobal KBAR %1            >sc.inp
echo  $setglobal LBAR %2            >>sc.inp

call gams mod_2x2.gms
```

By typing *run_sc 30 50* on the DOS prompt we effect that the following two lines are written to the file *sc.inp* is written.

```
$setglobal KBAR 30
$setglobal LBAR 50
```

The file *sc.inp* can be included within the core model code by means of the $INCLUDE statement. There we can use *KBAR* and *LBAR* as global variables and assign their values to the respective SCALARS.

→ *Version of 2x2x1_MODEL employing parameterization through global variables*

```
$TITLE: 2x2x1_MODEL

*       Set declaration and assignment
SETS
        I               goods /1 good_1, 2 good_2/;

$INCLUDE sc.inp

*       Data declaration and assignment
PARAMETERS
```

```
        A(I)            efficiency parameter /1 1.96, 2 1.889 /,
        ALPHA(I)        capital value share  /1 0.625, 2 0.667/,

        BETA(I)         expenditure share;

SCALARS
        KBAR            capital endowment    /%KBAR%/,
        LBAR            labor endowment      /%LBAR%/;
```