

Introduction to GAMS Software

A Manual for CGE Modelers

Jean-Christophe Dumont

Véronique Robichaud

February 2000

I. Introduction

The GAMS software (General Algebraic Modeling System) was developed by a group of economists from the World Bank in order to facilitate the resolution of large and complex non linear models on personal computer. As a matter of fact, GAMS allows to solve simultaneous non linear equation system, with or without optimization of some objective function.

(i) **Simplicity** (of implementation), (ii) **portability and transferability** (between users and systems) and (iii) **easiness of technical update** (because of the constant inclusion of new algorithms) are the main advantages of GAMS.

The seminal GAMS system was file oriented. The program must be created in ASCII format with any one of the usual text editor run by a DOS command.

The recent development of GAMS-IDE interface makes it even easier to use. GAMS-IDE works as a general text editor (WINDOWS 95 compatible) with the ability to launch and monitor the compilation and execution of typical GAMS programs. Besides, a GAMS sub-module has been recently developed which greatly facilitates exchanges with EXCEL and other spreadsheet software.

In this introduction note we will present the general structure of the GAMS program (Section II), followed by a detailed illustration (Section III). The output file will be described in Section IV and the GAMS-IDE software and new GAMS-Excel interface in Section V.

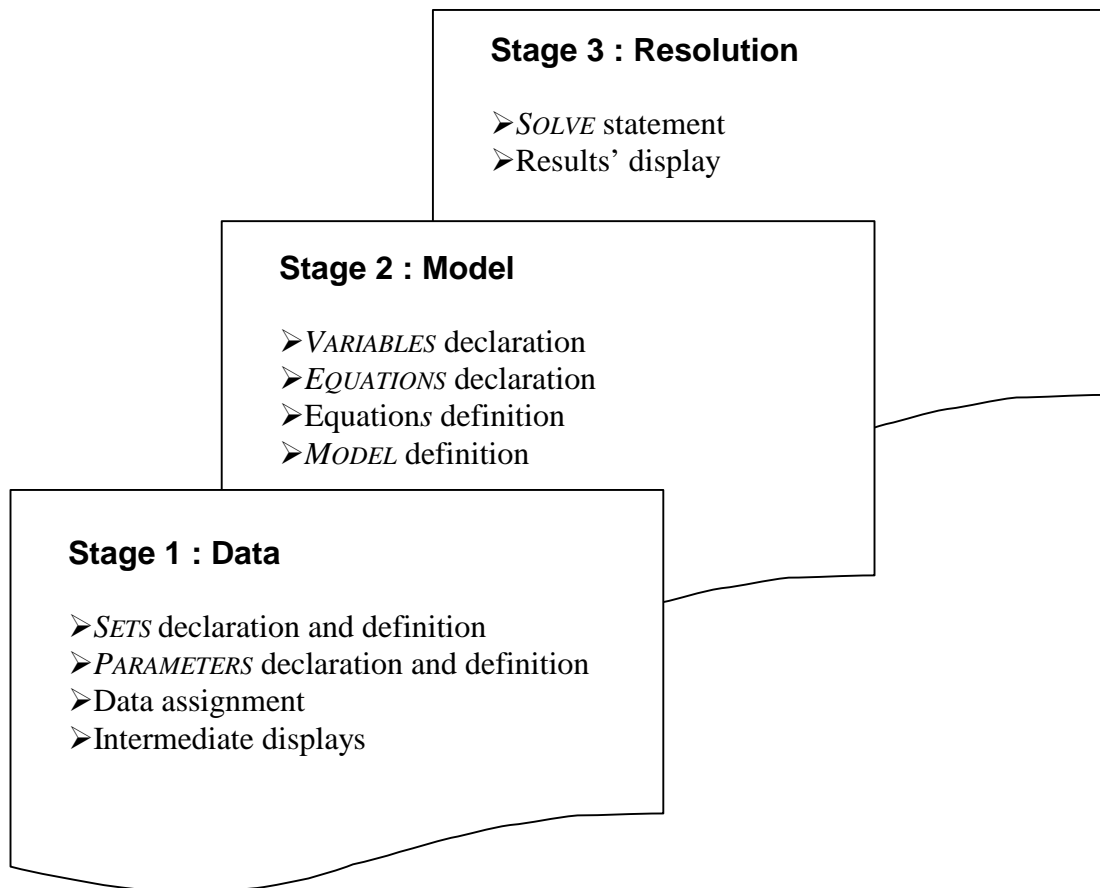
II. Architecture of GAMS programming : input file, resolution procedure and output settings.

1) The structure of the input GAMS Code

Typically, a Computable General Equilibrium (CGE) Model programmed in GAMS can be decomposed in 3 modules corresponding respectively to data entry (Stage 1), model specification (Stage 2) and solve procedure (Stage 3). The following chart gives an overall illustration of the structure of the GAMS syntax (the standard GAMS key words are typed in bold).

It is important to note that declaration and definition (or assignment) must be completed for each of the elements in use in the model (i.e. sets, parameters, variables or equations).

Organization chart of a typical Gams Code for CGE Models



Some general rules must be set apart concerning GAMS programming:

- ★ As a general rule, it is necessary to proceed to the declaration before to use any element. Sets and parameters must be declared at the very beginning of the program.

- ★ Even if it is not always necessary, a good habit is to end every statement by a semicolon in order to avoid unexpected compilation error.
- ★ GAMS allows for statement on several lines or several statement on the same line. This property can help to reduce the length of the code or facilitate printing.
- ★ Capital and small letters are not distinguished in GAMS.
- ★ Names for Set, Parameter, Variable, Equation or Model must start with a letter and can not contain more than 9 characters. GAMS key words (see infra) can not be used to declare an element.
- ★ Finally the main mathematical functions are described bellow:

Multiplication	*	Equality in an operation	=	Logarithm	<i>LOG(.)</i>
Subtraction	-	Summation	<i>SUM(set domain, element)</i>	Maximum	<i>MAX(.,.)</i>
Addition	+	Product	<i>PROD(set domain, element)</i>	Minimum	<i>MIN(.,.)</i>
Division	/	Absolute Value	<i>ABS(.)</i>		
Exponent	**	Exponential	<i>EXP(.)</i>		

2) Resolution procedure and output settings

Once the GAMS code has been written in a proper manner it must be saved. The name of the file must meet the standard DOS constraints (i.e. maximum 8 characters without symbols) and the default extension is *'.gms'*.

The Run command in the file menu (F9) of GAMS-IDE has then to be used (or the following DOS command has to be called for in the relevant folder: *'GAMS name.gms'*).

The output file created by this process has the same prefix as the initial program but get the extension *'.lst'*. It contains either the initial program with compilation error identification if any, either the default output settings as described bellow.

Output in the file <i>'name.lst'</i>
➤ Echo print of the initial program
➤ Equation listing
➤ Column listing
➤ Model statistics
➤ Solve summary
➤ Results

The next two sections will give an illustration of the content and the nomenclature of GAMS programming. The example is taken from a small theoretical CGE Model (Model 0).

II. Writing a GAMS Program

All statements previously presented are used in a complete GAMS program and some additional options are also introduced. For further information on any of these commands, the reader should refer to the GAMS User's Guide¹.

1) *The Calibration*

The first part of the modeling process consists in entering data, which will be referred as benchmark data, and evaluating parameters consistent with these values. This process is called the *calibration*. In the case of CGE modeling, the benchmark data are usually drawn from the Social Accounting Matrix (SAM). Parameters that can be calibrated will be determined with those.

i) Title option

Although not necessary, the title option allows a more comprehensible output listing. The text following the \$TITLE command will appear as a header on every page of the output listing. A sub-title can also be added to this header using the command \$STITLE.

```
$TITLE                                THEORETICAL MODEL
$STITLE                                CAPITAL IMMOBILE ACROSS SECTORS

*Model of a closed economy producing 2 goods using 2 factors
*Manila February March 2000
```

In this example, THEORETICAL MODEL will appear as the first line of each page in the output file and CAPITAL IMMOBILE ACROSS SECTORS on the second one. The two other lines, are comments that will not appear in the output. GAMS will not read lines beginning with an asterisk (*). The use of comments can be useful to the modeler in order to organize its program in a comprehensive manner. An other manner to do this would be to put the comment between the \$ONTEXT and \$OFFTEXT command.

ii) Set definition

The definition of sets is useful for multidimensional variables. It corresponds to the indexes in mathematical representations of models.

```
SET I sectors                        /GOOD    agriculture and industry
                                      SER      services/
```

The SET statement defines the set of sectors, denoted by the letter I, whom elements are GOOD and SER. After each symbol (I, GOOD and SER) a description can be added.

iii) Parameters definition

The parameters should now be defined. Parameters are the elements in the model's equations that will not change after a simulation, such as elasticities, tax rates, distribution and scale coefficients. In addition to these parameters, benchmark variables

¹ Brooke A., D. Kendrick & A. Meeraus (1998), A. GAMS. A user's Guide. The Scientific Press, San Fransisco.

are also defined for their value at the base year will not change after a simulation. A common way to define these variables is to add the letter "O" after the variable name so it will not be confused with the 'true' variable. Parameters and benchmark variables definition begins with the statement `PARAMETER` and end with a semicolon. Once again, it is useful to put a description after the parameter designation, as it is done in the example bellow. When a parameter is subject to an index, like A_i , the set over which it is defined is put between parentheses, $A(I)$.

```
PARAMETER

  A(I)           Scale coefficient (Cobb-Douglas production function)
  alpha(I)      Share (Cobb-Douglas production function)
  delta         Share parameter (CES utility function)
  sigma         Substitution elasticity (CES utility function)

*Prices
  wo           Wage rate
  ro(I)        Rate of return to capital in sector I
  PO(I)        Price of good I

*Production
  XSO(I)       Production of sector I

*Factors
  KDO(I)       Sector I demand for capital
  LSO          Total labour supply
  LDO(I)       Sector I demand for labour

*Demand
  CO(I)        Consumption of good I

*Income
  YHO          Household income
;
```

iv) Data handling

Once the sets, parameters and benchmark variables are defined, data must be entered. This can be done using the `TABLE` command, which is useful for multidimensional variables. The syntax in the following: `TABLE name(row domain, column domain) description`. Here is an example:

```
TABLE DP(*,I) Miscellaneous data by sector

          GOOD          SER
XSO       26100.0       21260.0
LDO       13320.0       15540.0
KDO       12780.0        5720.0
CO        26100.0       21260.0
PO         1.0          0.85
ro        1.25         1.25
;
```

The table is named DP, is defined over the set I in column and over various variables in row, which is represented with the asterisk (*). At the beginning of each following row is the name of the variable followed by its value for each element of set I, that is GOOD and SER placed above each column. Values can be expressed without or with decimals. A semicolon indicates the end of the table.

Values for variables and parameters with one dimension can be entered between slashes using the SCALAR statement, as it is done below:

```
SCALAR

YHO          / 47360.0/
wo           / 0.75/
sigma        / 0.8 /
;
```

When data are entered in a table, it then has to be assigned to the relevant variable. In the example, the value of XSO(I) can be found in the table DP at the intersection of the row named XSO and the column I.

```
XSO(I)       = DP("XSO",I);
LDO(I)       = DP("LDO",I);
KDO(I)       = DP("KDO",I);
CO(I)        = DP("CO",I);
PO(I)        = DP("PO",I);
ro(I)        = DP("ro",I);
```

v) Calculation of variables and calibration

Other variables and parameters can be calculated or calibrated from the ones entered previously. At the end of each calculation, a semicolon is put. The sequence in which they are calculated is very important because GAMS will use the last value assigned to a variable or a parameter.

```
*Volume calculations
LDO(I)       = LDO(I)/wo;
KDO(I)       = KDO(I)/ro(I);
XSO(I)       = XSO(I)/PO(I);
LSO          = SUM(I,LDO(I));
CO(I)        = CO(I)/PO(I) ;

*Production (Cobb-Douglas)
alpha(I)     = wo*LDO(I)/(PO(I)*XSO(I));
A(I)         = XSO(I)/((LDO(I)**alpha(I))*(KDO(I)**(1-alpha(I))));

*Consumption (CES utility function)
delta        = (PO("GOOD")/PO("SER"))*(CO("GOOD")
              /CO("SER"))**(1/sigma))/
              (1+PO("GOOD")/PO("SER")*
              (CO("GOOD")/CO("SER"))**(1/sigma));
```

vi) Display of parameters and variables

In the output listing, values of parameters are not automatically printed. The `DISPLAY` option will ask GAMS to include them in the output file. Note that the names of parameters and benchmark variables are entered without dimension.

```
DISPLAY          sigma,delta,alpha,A;
```

2) *The Model*

i) Variable declaration

All variables appearing in the equations must first be declared. The statement `VARIABLES` begins this procedure and ends with a semicolon. Following the variable name, for example `w`, a brief description can be added. As in the calibration process, the variables in the following example are grouped in categories for comprehensive purpose only.

```
VARIABLES

*Price
  w           Wage rate
  r(I)        Rate of return to capital in sector I
  P(I)        Price of good I

*Production
  XS(I)       Production of sector I

*Factors
  KD(I)       Sector I demand for capital
  LD(I)       Sector I demand for labour
  LS          Total labour supply

*Demand
  C(I)        Consumption of good I (volume)

*Income
  YH          Household income

*Equilibrium
  LEON        Walras law verification variable
;
```

ii) Equations declaration

As with the previous component of the model, each of the equation must be declared and defined. This step begins with the `EQUATIONS` statement followed by the symbols for which a description can be added. For example, the equation named `SUPPLY(I)` is followed by a short description. When all equations are declared, a semicolon indicates the end.

```
EQUATIONS

*Production
  SUPPLY(I)   Production function for sector I
```

LDEM(I)	Labour demand for sector I
*Income	
INCH	Household income
*Demand	
DGOOD	Demand for goods
DSERV	Demand for services
*Prices	
RETK(I)	Capital remuneration from sector I
*Equilibrium	
DOMABS	Domestic absorption (goods)
LEQUI	Labour market equilibrium
*Other	
WALRAS	Verification of Walras law
;	

iii) Equations assignment

The syntax for equation definition is the following. First the name and dimension is stated followed by two dots. Then the equation is defined and ends with a semicolon. The =E= between the left hand side and the right hand side of the equation simply means "equals to" as opposed to =G= or =L= which respectively represents "greater than" and "less than". Each equation can only be defined once but the variables can appear in several equations and on both sides of the same equation. Here, the order in which the equations are defined does not matter since all equations are solved simultaneously.

```

SUPPLY(I)..      XS(I)   =E=   A(I)*LD(I)**alpha(I)*KD(I)**(1-alpha(I));
LDEM(I)..       LD(I)   =E=   P(I)*alpha(I)*XS(I)/w;
INCH..          YH      =E=   SUM(I,w*LD(I))+SUM(I,r(I)*KD(I));
DGOOD..         C("SER")=E=   YH/(P("SER")+P("GOOD")*(P("SER")/
                               P("GOOD")*delta/(1-delta)**sigma);
DSERV..         C("GOOD")=E=  YH/(P("GOOD")+P("SER")*(P("GOOD")/
                               P("SER")*(1-delta)/delta)**sigma);
RETK(I)..       r(I)    =E=   (P(I)*XS(I)-w*LD(I))/KD(I);
DOMABS..        XS("SER")=E=  C("SER");
LEQUI..         LS      =E=   SUM(I,LD(I));
WALRAS..        LEON    =E=   XS("GOOD")-C("GOOD");

```

iv) Initialization procedure

For all variables declared previously, a value must be assigned. In order to resolve the equations, GAMS has to start from a benchmark value. If no value is associated to a variable, GAMS will use zero, which could cause problems if the variable is at the

denominator of a division. For the endogenous variables, that is the ones whom values will be affected by a simulation, the suffix `.L` (for level) is used as follow:

```
XS.L(I)      = XSO(I);
LD.L(I)      = LDO(I);
KD.L(I)      = KDO(I);
w.L          = wo;
P.L(I)       = PO(I);
r.L(I)       = ro(I);
YH.L        = YHO;
C.L(I)       = CO(I);
```

For the exogenous variables, that is the ones whom values will not be affected by a simulation, the suffix `.FX` (for fixed) is used:

```
*P(GOOD) is the numeraire, capital immobile across sectors
* total labour supply is exogeneous

P.FX("GOOD") = PO("GOOD");
LS.FX = LSO;
KD.FX("GOOD") =KDO("GOOD");
KD.FX("SER")=KDO("SER");
```

In both cases, the syntax is very similar; that is, the name of the variable, the suffix, the variable dimension, the equal sign (=), the benchmark value defined in the calibration process and a semicolon.

v) Model execution

Finally, two lines must be added in order to solve the equation system. The first one defines the model. It begins with the `MODEL` statement, followed by a name (`THEORY` in the example), a brief description (`THEORETICAL MODEL`), the list of equations to be resolved between slashes and a semicolon. When all equations are to be resolved, the key word `ALL` is put instead of the complete list of equations.

```
MODEL THEORY THEORETICAL MODEL /ALL/;
```

The second line defines the procedure to be used in resolving the model. The `SOLVE` statement is followed by the name of the model, `USING` and the procedure to be used to resolve the model. In the example, the `MCP` (Mixed Complementarity Problems) procedure is used. The procedure to be used is determined by the type of model to solve. For example, linear programs can be solved using the `LP` procedure, while non-linear programs can be solved using `NLP`. Each procedure uses a different solver (`MINOS`, `CONOPT`, `MILES`, ...). These solvers use different algorithms that can be efficient in resolving some system but less in other cases. In particular, the `NLP` procedure (which could have been used in the present example) like most procedures, requires the optimization of an objective variable.

```
SOLVE THEORY USING MCP;
```

III. Reading a GAMS output file

Once GAMS has resolved an equation system, it creates an output file. In this section the output file of the program presented in the previous section is described.

1) *Echo print*

The first part of the output file is simply the reproduction of the model with line numbers on the left hand side, for further references in the file. If there are errors in the program, this is the only part of the output that will appear. Here is an example. There is a parenthesis missing at the end of the calculation of the parameter `alpha`. In the echo print of the initial program, GAMS will put four asterisks (****) below the line where a mistake has been detected, followed by a dollar sign (\$) and a number corresponding to a specific error.

```
95 alpha(I) = wo*LDO(I)/(PO(I)*XSO(I);
****                                     $8
96 A(I)      = XSO(I)/((LDO(I)**alpha(I))*(KDO(I)**(1-alpha(I)))));
```

Because of this error, GAMS was unable to solve the model, thus another error is put on the last line of the program.

```
230
231 MODEL THEORY THEORITICAL MODEL /ALL/;
232 SOLVE THEORY USING MCP;
****                                     $257
```

At the end of the program echo print, a description of the error code is given in order to facilitate the correction:

```
Error Messages
 8  ')' expected
257 Solve statement not checked because of previous errors
```

When no errors are encountered, the following outputs also figure in the listing.

2) *Displayed parameters*

In the calibration process, the `DISPLAY` statement was used in order to print the value of some parameters in the output file. These values appear after the echo print as follow:

```
---- 105 PARAMETER SIGMA = 0.800 Substitution elasticity (CES utility function)
      PARAMETER DELTA = 0.554 Share parameter (CES utility function)

---- 105 PARAMETER ALPHA      Share (Cobb-Douglas production function)

GOOD 0.510,    SER 0.731

---- 105 PARAMETER A          Scale coefficient (Cobb-Douglas production function)

GOOD 1.926,    SER 1.812
```

3) Equation listing

This section presents, for each equation, the specific instance of the model that is created when actual values of the sets and parameters are used. At the end of the equation, the value of the left hand side is put between parenthesis (LHS= " "). When the value of the left hand side differs from the one on the right hand side, GAMS add three asterisks (***). This indication can be useful in finding problems linked to incorrect parameter calibration or improper variable initialization.

```
---- SUPPLY      =E= Production function for sector I

SUPPLY(GOOD)..  XS(GOOD) - (1.25)*KD(GOOD) - (0.75)*LD(GOOD) =E= 0 ; (LHS = 0)

SUPPLY(SER)..  XS(SER) - (1.4706)*KD(SER) - (0.8824)*LD(SER) =E= 0 ; (LHS = 0)

---- LDEM        =E= Labour demand for sector I

LDEM(GOOD)..  (23680)*W-(17760)*P(GOOD)-(0.6805)*XS(GOOD)+LD(GOOD) =E= 0 ; (LHS = 0)

LDEM(SER)..  (27626.6667)*W-(24376.4706)*P(SER)-(0.8284)*XS(SER)+LD(SER) =E= 0 ; (LHS=0)
```

4) Column listing

Following the equation listing, GAMS prints the equivalent information by column instead of row. For each variable, the lower and upper bounds, the level calculated by the solver and the value of the coefficient affecting the actual variable in each equation are given. If the coefficient is non-linear, it is put between parentheses. In the example below, the wage (noted w) has no lower bounds (thus $-INF$ for minus infinity), no upper bounds ($+INF$) and the level that satisfies equilibrium is 0.75. The variable appears in five equations, $LDEM(I)$, $INCH$ and $RETK(I)$, which are non-linear, so the value of the coefficient affecting w is put between parentheses.

```
Column Listing      SOLVE THEORY USING MCP FROM LINE 231

---- W            Wage rate
W
      (.LO, .L, .UP = -INF, 0.75, +INF)
      (23680)      LDEM(GOOD)
      (27626.6667) LDEM(SER)
      (-38480)     INCH
      (1.7371)     RETK(GOOD)
      (4.528)      RETK(SER)

---- R            Rate of return to capital in sector I
R(GOOD)
      (.LO, .L, .UP = -INF, 1.25, +INF)
      (-10224)     INCH
      1            RETK(GOOD)

R(SER)
      (.LO, .L, .UP = -INF, 1.25, +INF)
      (-4576)     INCH
      1            RETK(SER)
```

5) Model Statistics

The purpose of this section is to provide information about the size and non-linearity of the model. The `BLOCK` gives the number of equations and variables in the model (in the example, 9 and 10 respectively) while `SINGLE` gives the number of rows and columns generated by the solver (here 12 and 16). The `NON-ZERO ELEMENTS` count refers to the number of non-zero coefficients. The four remaining entries gives information about the non-linearity of the model. The `NON LINEAR N-Z` indicates the number of nonlinear matrix entries in the model. The `CODE LENGH` and the `CONSTANT POOL` give the user the level of complexity on the non-linearity. Finally, this section indicates the time used to generate the model (`GENERATION TIME`).

MODEL STATISTICS

BLOCKS OF EQUATIONS	9	SINGLE EQUATIONS	12
BLOCKS OF VARIABLES	10	SINGLE VARIABLES	16
NON ZERO ELEMENTS	50	NON LINEAR N-Z	33
DERIVATIVE POOL	10	CONSTANT POOL	17
CODE LENGTH	530		

GENERATION TIME = 0.000 SECONDS 1.4 Mb WIN-18-100

6) Solve Summary

While all previous information referred to the solving process, the followings concern the solution itself. The first section on this matter is the solve summary. It recalls the name of the model (here `THEORY`), the type of model (`MCP`) and the solver used (`MILES`). The `SOLVER STATUS` indicates the way the solver terminated. If the resolution has been interrupted by the user or by the solver (because of errors), this is where it will be indicated. In our example, the solver terminated in a normal way, which is indicated by `1 NORMAL COMPLETION`. While the `SOLVER STATUS` refers to the resolution process, the `MODEL STATUS` refers to the solution itself. The code `1 OPTIMAL` indicates that the solution is optimal. The `RESOURCE USAGE, LIMIT`, gives information about the time, in seconds, taken by the solver and the upper limit allowed for the solver. The default upper bound is 1000 but can be changed by the user. The `ITERATION COUNT, LIMIT` provide the number of iterations used by the solver and the upper limit (default being 1000). Finally, `EVALUATION ERRORS` indicates the number of errors encountered by the solver and the maximum error allowed (default being 0).

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

MODEL	THEORY		
TYPE	MCP		
SOLVER	MILES	FROM LINE	231
**** SOLVER STATUS	1	NORMAL COMPLETION	
**** MODEL STATUS	1	OPTIMAL	
RESOURCE USAGE, LIMIT	0.000	1000.000	
ITERATION COUNT, LIMIT	0	10000	
EVALUATION ERRORS	0	0	

7) Solution Listing

In this section, GAMS lists the solution found by the solver row-by-row then column-by-column. For each equation and each variable it indicates the lower and upper bounds, the level and the marginal. While the bounds are defined by the modelers, the level and the marginal are calculated by the solver. The dot "." means zero. Here is an example of the equation list:

```

---- EQU SUPPLY          Production function for sector I
      LOWER    LEVEL    UPPER    MARGINAL
GOOD      .      .      .      .
SER       .      .      .      .

---- EQU LDEM           Labour demand for sector I
      LOWER    LEVEL    UPPER    MARGINAL
GOOD      .      .      .      .
SER       .      .      .      .

```

And here is an example of the variable listing:

```

      LOWER    LEVEL    UPPER    MARGINAL
---- VAR W              -INF      0.750    +INF      .
W              Wage rate

---- VAR R              Rate of return to capital in sector I
      LOWER    LEVEL    UPPER    MARGINAL
GOOD    -INF    1.250    +INF      .
SER     -INF    1.250    +INF      .

```

8) Report Summary

The report summary section indicates the count of rows or columns that GAMS marked with the following remarks and the count of errors.

```

**** REPORT SUMMARY :
                   0      NONOPT
                   0      INFEASIBLE
                   0      UNBOUNDED
                   0      REDEFINED
                   0      ERRORS

```

9) File Summary

The last part of the output file indicates the names of the input and output files. In the following example, the input file name is OKIMM.GMS and the output file is OKIMM.LST and files are located in different directories.

```
**** FILE SUMMARY

INPUT      C:\MY DOCUMENTS\MIMAP\FORMATION MANILLE\OKIMM.GMS
OUTPUT     C:\WINDOWS\GAMSDIR\OKIMM.LST
```

IV. GAMS Interfaces for Windows

1) The Gams-IDE Interface

GAMS-IDE has been developed in order to facilitate the use of GAMS in a Windows environment. It acts as a general text editor with the ability to launch and monitor compilation and execution of GAMS Models. Because of the windows environment several models can be run at the same time and other programs can also be running during GAMS completion.

GAMS-IDE software appears with two windows, one is the EDIT WINDOW, organized as a tabbed notebook, which includes the current programs and output files and the other is the PROCESS WINDOW in which the completion status of the last runs are shown.

One of the significant advantage of GAMS-IDE interface is that it allows to slide very easily from the program file to the listing file. Compilation errors appear in red in the PROCESS WINDOW. A double-click with the mouse will open the *gms* file and position the cursor on the error. If you want to see the error in the listing file, hold down the Shift key while double-clicking the red line. Important locations in the listing file are shown in blue. A double click on such a line will position the cursor in the listing file. A double-click on a line which is not red or blue, will position the cursor in the listing file.

Another advantage of the GAMS-IDE interface relies in the project file concept. It serves to restore the EDIT WINDOW in the state they were when the project was closed for the last time. In addition, the directory where the project file is located is used as the default directory for file operations when GAMS executes, such as searching for INCLUDE files and writing the PUT files. You can create or open a project through the file Menu. The name of the current project appears in the GAMS-IDE header.

The EDIT WINDOW acts as standard editor. Some of the main editor keys are described in the table bellow. They can be run through the Edit Menu.

You can use the function search or replace from the Search Menu to look for a work in any of the windows. GAMS-IDE also permits to find matching parenthesis (bracket,

curly). This feature is very useful in order to check the consistence of the writing of the GAMS program. It can be run from the Search Menu or using the key F8. The model can then be run either using the mouse and click on the run button on the main window, either from the Run Command of File Menu or using the F9 key.

Control	Command
Ctrl+Ins or Ctrl+C	Copy the selected text to the clipboard
Shift+Del or Ctrl+X	Delete the selected text after copying it to the clipboard
Ctrl+Y	Delete the current line
Ctrl+T	Delete the word to the right of the caret
Shift+Ins or Ctrl+V	Paste the text from the clipboard
Ctrl+Shift+Z	Redo the last undone operation
Ctrl+Z	Undo the last operation
Ctrl+Left or Right	Move the caret left (or Right) one word

2) The GAMS-Excel Interface²

This module has been programmed by A. Malivev. It is not a software in itself but rather an add-on to GAMS 2.5, called XLLINK.

The XLLINK system includes two archives, [inclub.pck](#) and [xllink.exe](#). The first contains general data management routines used by XLLINK, and the second is the XLLINK program file, a console application. To install the system, copy inclub.pck and xllink.exe to the GAMS system directory and run GAMSINST.

Three "libinclude" calls are provided, which make it possible to transfer data between GAMS programs and Excel workbooks. Briefly the interface routines are as follows (see T. Rutherford & A. Malivev –1999) :

➤XLIMPORT reads data from spreadsheets or workbooks *at compile time*. It can be used to retrieve data from any worksheet format which can be imported by Excel. The spreadsheet range must be specified explicitly.

```
$LIBINCLUDE XLIMPORT parameter wbfile range
```

```
$LIBINCLUDE XLIMPORT A A.WK1 A1:D20
```

- Generate an error if A.WK1 does not exist.
- Retrieve data from range A1:D20 in parameter statement for A.
- Generate a GAMS error if there is a domain violation.

² Cf Rutherford, T.F and A. Maliyev (July 1999), The GAMS-Excel Interface, Miméo Department of Economics, University of Colorado for further information.

➤XLDUMP writes data and labels into an existing Excel workbook. You can specify a worksheet within the workbook into which the data is to be written. Row and column order are defined by the GAMS program symbol table as in a GAMS DISPLAY statement.

```
$LIBINCLUDE XLDUMP ident[.suffix] wfile[.xls] [worksheet]
```

```
$LIBINCLUDE XLDUMP A REPORT.XLS Sheet2
```

- A new workbook is created when wfile does not exist but if a new file is to be created, the ".xls" extension must be specified.
- The data is written to the upper left corner of the target worksheet.
- Data are written to the default worksheet of an existing workbook if no target worksheet is specified.

➤XLEXPOR writes data into spreadsheets according to the row and column labels in the specified target range. The target range is always explicit (static), and we only transfer data for which labels are specified.

```
$LIBINCLUDE XLEXPOR ident[.suffix] sfile range [/m]
```

```
$LIBINCLUDE XLEXPOR A OUTPUT.XLS Solution!A1:D20
```

- Erase the target range A1:D20.
- Copy nonzero elements of parameter A for which matching labels are found in target range A1:D20 of the worksheet named "Solution".
- Do not transfer data for unspecified labels (only requested data is inserted).

Please note that, in each case, (i) the parameter to be imported or exported must be declared and dimensioned prior to the import; (ii) data transfer occurs at execution time and (iii) by default, zero values are not transferred. If you need to move zeros into the spreadsheet cells, assign values of EPS.