AN INTRODUCTION TO GAMS[1]

VÉRONIQUE ROBICHAUD, JANUARY 2017

Introduction

The GAMS (*General Algebraic Modeling System*) software was originally developed by a group of economists from the World Bank to facilitate the resolution of large and complex non-linear models on personal computers. In fact, GAMS allows us to solve simultaneous non-linear equation systems, with or without the optimization of some objective functions.

The main advantages of GAMS are (i) **simplicity** of implementation, (ii) **portability and transferability** between users and systems, and (iii) **ease of technical updates** thanks to the constant inclusion of new algorithms.

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

The development of the GAMS-IDE interface in the late 1990s makes it even easier to use. GAMS-IDE works as a general text editor compatible with WINDOWS and allows us to launch and monitor the compilation and execution of typical GAMS programs. In this introduction, we will present the general structure of the GAMS program followed by a detailed illustration including a description of the output file.

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

The structure of the input GAMS Code

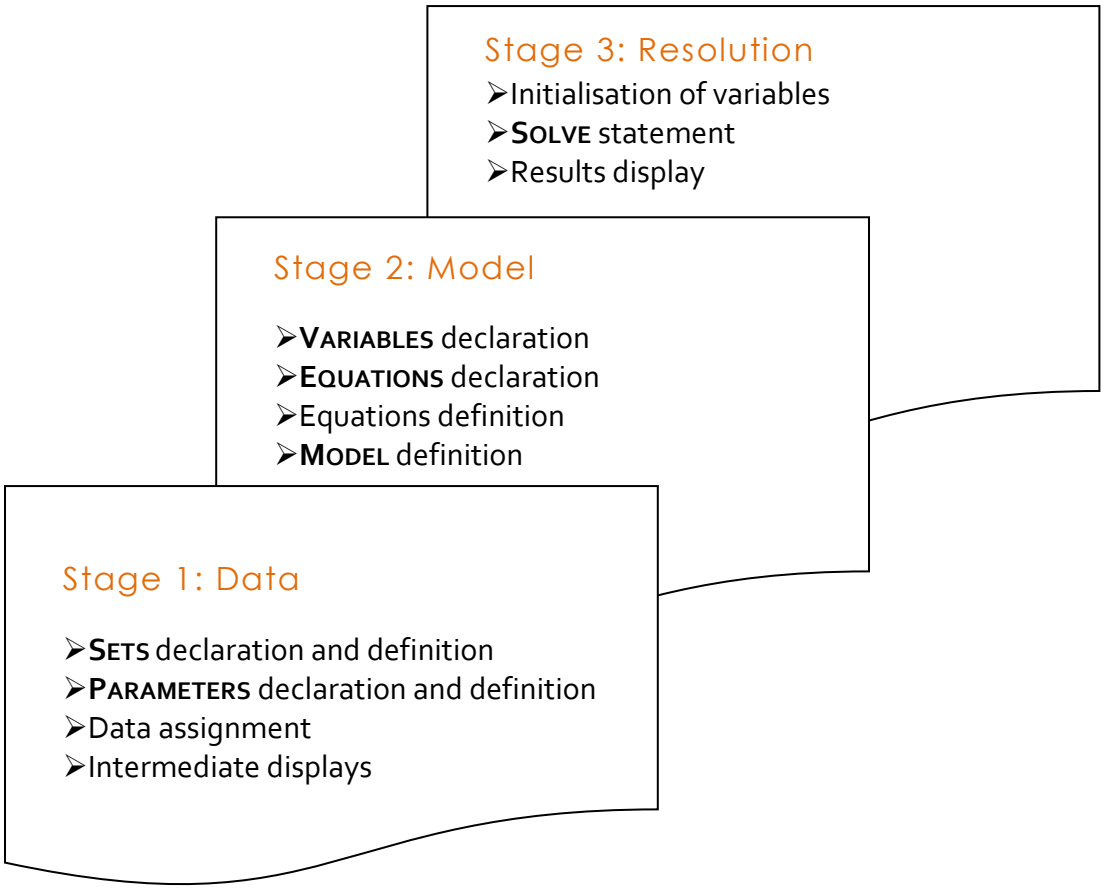
Typically, a computable general equilibrium (CGE) model programmed in GAMS can be broken down to three modules: data entry, model specification and resolution (see Figure 1 below).

---

[1] This document is an extract from "GAMS an Introduction", a pedagogical paper written by Jean-Christophe Dumont and Véronique Robichaud in 2000.

The chart gives an overall illustration of the structure of the GAMS syntax. Note that the standard GAMS keywords appear in bold.

Figure 1: Organization chart of a typical GAMS code for CGE models



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

Some general rules to follow during GAMS programming:

- As a general rule, it is necessary to complete the declaration of any element before using it. In particular, sets 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 with a semicolon in order to avoid unexpected compilation errors.
- Capital and small letters are not distinguished in GAMS.
- Finally, the main mathematical functions are described below:

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

Resolution procedure and output settings

Once the GAMS code has been written in a proper manner it must be saved with the extension **.gms**. To run the code, you can either choose the run command in the file menu, press F9, or click on the red arrow button.

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

Output in the file 'name.lst'

- Echo print of the initial program
- Equation listing
- Column listing
- Model statistics
- Solve summary
- Results

The next two sections will illustrate the content and terminology of GAMS programming. The example is taken from an adaptation of the model AUTA presented in: Decaluwé, B., A. Martens and L. Savard (2001), "La politique économique du développement et les modèles d'équilibre général calculable. Une introduction", Montreal, Presses de l'Université de Montréal, p.524.

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, please refer to the GAMS User's Guide[2] available under the help menu.

Calibration of a CGE model

The first part of the modeling process consists of entering data, which will be referred to 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 is usually drawn from the Social Accounting Matrix (SAM). Parameters that can be calibrated will be determined by the benchmark data.

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**.

In the example below, `MODEL AUTA` will appear as the first line of each page in the output file and `AUTARKY WITHOUT GOVERNMENT` as the beginning of the second line. The following lines are comments; GAMS will not read lines beginning with an asterisk (*). The use of comments can be useful to the modeller in order to organize his/her program in a comprehensive manner. Another way to insert comments is to put the text between the **$ONTEXT** and **$OFFTEXT** command.

```
$TITLE    MODEL AUTA
$STITLE   AUTARKY WITHOUT GOVERNMENT

* Model of a closed economy without government producing 3 goods using
* 2 factors owned by 2 types of households.
```

[2] GAMS — A User's Guide. Tutorial by Richard E. Rosenthal c 2006. GAMS Development Corporation, Washington, DC, USA.

Set definition

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

```
SETS I Industries and commodities
/ AGR agriculture
  MAN manufacturing
  SER services /

BNS(I) Goods
/ AGR agriculture
  MAN manufacturing /

H Households
/ SAL labour endowed households
  CAP capital endowed households /

ALIAS (i,j)
;
```

The SETS statement defines the sets of the model. In this example, three sets are defined. The first one, denoted by the letter I, contains elements are AGR, MAN and SER. After each symbol (AGR, MAN and SER) a description can be added. The elements of a set are placed between slashes (/). The second set, named BNS, is in fact a subset of I, indicated by the (I) part and includes only the two first elements. The last set represents households. A final command, ALIAS, is used to allow the modeller to use a second index (J) that represents exactly the same elements as set I. This is useful when using multi-indexed variables.

Parameters definition

The parameters should now be defined. Parameters are the elements in the equations that will not change after a simulation, such as elasticity, tax rates, distribution and scale coefficients. In addition to these parameters, benchmark variables are also defined by their value at the base year and will not change after simulation. A common way to define these variables is to add an "O" after the variable name so it will not be confused with the 'true' variable. Parameter and benchmark variable definition begins with the statement PARAMETERS and ends with a semicolon. Once again, it is useful to include a description after the parameter designation, as in the example. When a parameter is subject to an index, such as A_j, the set over which it is defined is written between parentheses, A(j).

```
PARAMETERS

 A(j)             Scale parameter (Cobb-Douglas - production function)
 aij(i,j)         Coefficient (Leontief - intermediate consumption)
 alpha(j)         Elasticity (Cobb-Douglas - production function)
 gamma(i,h)       Share of commodity i in type h household consumption budget
 io(j)            Coefficient (Leontief - total intermediate consumption)
 lambda           Share of capital income received by capitalists
 mu(i)            Share of commodity i in total investment expenditures
 psi(h)           Average propensity to save of type h household
 v(j)             Coefficient (Leontief - value added)

* Definition of variables for the base year
* Volume variables (quantities)
 CO(i,h)          Consumption of commodity i by type h households
 CIO(j)           Total intermediate consumption of industry j
 DIO(i,j)         Intermediate consumption of commodity i in industry j
 DITO(i)          Total intermediate demand for commodity i
 INVO(i)          Final demand of commodity i for investment purposes
 KDO(j)           Industry j demand for capital
 KSO(j)           Capital supply in industry j
 LDO(j)           Industry j demand for labour
 LSO              Total labour supply
 VAO(j)           Value added of industry j
 XSO(j)           Output of industry j
* Prices
 PO(i)            Price of commodity i
 PCIO(j)          Intermediate consumption price index of industry j
 PVAO(j)          Price of industry j value added
 RO(j)            Rental rate of capital in industry j
 WO               Wage rate

* Nominal variables (values)
 CTHO(h)          Consumption budget of type h households
 DIVO             Dividends
 ITO              Total investment
 SFO              Business savings
 SHO(h)           Savings of type h households
 YFO              Business income
 YHO(h)           Income of type h households
;
```

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 is as follows: `TABLE` *name*(*row domain*, *column domain*) *description*. For example:

```
TABLE SAM(*,*) Social accounting matrix

          LD       KD      SAL      CAP        F      AGR      MAN      SER      ACC
LD                                                    300      100      200
KD                                                    100      150      100
SAL       600
CAP                210                          70
F                  140
AGR                          162       21              50      150       90       27
MAN                          108       84              20      150       90      173
SER                          270      105              30       75      120
ACC                           60       70        70
TOT       600      350      600      280       140      500      625      600      200
;
```

The table is named `SAM` and has two dimensions. When the column/row labels do not refer to any specific set, an asterisk is used. Here, we have used the entire SAM but with the elements of the sets, when relevant, as the title of each column and row. This will greatly facilitate data assignment. Values can be expressed with or without decimals but must fall under only one column title. A semicolon indicates the end of the table.

When data is entered in a table, it then has to be assigned to the relevant variable. In the example, the value of `CO(i,h)` can be found in the table `SAM` at the intersection of the row named i and the column `h`. When referring to a row or a column that does not represent an element of a defined set, brackets are used. For example, the value of dividends (DIVO) is given in table SAM at the intersection of row 'CAP' and column 'F'.

```
* Assignment of variables
DIVO              = SAM('CAP','F');
ITO               = SAM('TOT','ACC');
SFO               = SAM('ACC','F');
SHO(h)            = SAM('ACC',h);
YFO               = SAM('TOT','F');
YHO(h)            = SAM('TOT',h);
XSO(j)            = SAM('TOT',j);
CO(i,h)           = SAM(i,h);
DIO(i,j)          = SAM(i,j);
INVO(i)           = SAM(i,'ACC');
KDO(j)            = SAM('KD',j);
LDO(j)            = SAM('LD',j);
```

Values for variables and parameters can also be entered simply as follows. In our example, all prices are set to one.

```
* Prices
WO                = 1;
RO(j)             = 1;
PO(i)             = 1;
```

Calculation of variables and calibration

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

```
* Computation of variables in volume
LDO(j)            = LDO(j)/WO;
KDO(j)            = KDO(j)/RO(j);
XSO(j)            = XSO(j)/PO(j);
CO(i,h)           = CO(i,h)/PO(i);
INVO(i)           = INVO(i)/PO(i);
DIO(i,j)          = DIO(i,j)/PO(i);

* Calibration of other variables
LSO               = SUM[i,LDO(i)];
KSO(j)            = KDO(j);
VAO(j)            = LDO(j)+KDO(j);
PVAO(j)           = {WO*LDO(j)+RO(j)*KDO(j)}/VAO(j);
DITO(i)           = SUM[j,DIO(i,j)];
CIO(j)            = SUM[i,DIO(i,j)];
PCIO(j)           = SUM[i,PO(i)*DIO(i,j)]/CIO(j);
CTHO(h)           = YHO(h)-SHO(h);

* Calibration of parameters
* Production (Cobb-Douglas and Leontief)
alpha(j)          = WO*LDO(j)/{PVAO(j)*VAO(j)};
A(j)              = VAO(j)/{LDO(j)**alpha(j)*KDO(j)**(1-alpha(j))};
v(j)              = VAO(j)/XSO(j);
io(j)             = CIO(j)/XSO(j);
aij(i,j)          = DIO(i,j)/CIO(j);

* Distribution parameters
gamma(i,h)        = PO(i)*CO(i,h)/CTHO(h);
lambda            = {YHO('cap')-DIVO}/SUM[j,RO(j)*KDO(j)];
mu(i)             = PO(i)*INVO(i)/ITO;
psi(h)            = SHO(h)/YHO(h);
```

Displaying parameters and variables

In the output listing, values of parameters are not automatically printed. Use the **DISPLAY** option to show these values. Note that the names of parameters and benchmark variables are entered without dimension.

```
* Parameters to be displayed in the output file
DISPLAY A, alpha, io, v, aij, gamma, psi, mu, lambda;
```

The model

Variable declaration

All variables, endogenous or exogenous, 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 optional description can be added. In the example below, we have grouped the variables by category.

```
VARIABLES
* Volume variables (quantities)
 C(i,h)          Consumption of commodity i by type h households
 CI(j)           Total intermediate consumption of industry j
 DI(i,j)         Intermediate consumption of commodity i in industry j
 DIT(i)          Total intermediate demand for commodity i
 INV(i)          Final demand of commodity i for investment purposes
 KD(j)           Industry j demand for capital
 KS(j)           Capital supply in industry j
 LD(j)           Industry j demand for labour
 LS              Total labour supply
 VA(j)           Value added of industry j
 XS(j)           Output of industry j
* Prices
 P(i)            Price of commodity i
 PCI(j)          Intermediate consumption price index of industry j
 PVA(j)          Price of industry j value added
 R(j)            Rental rate of capital in industry j
 W               Wage rate
* Nominal variables (values)
 CTH(h)          Consumption budget of type h households
 DIV             Dividends
 IT              Total investment
 SF              Business savings
 SH(h)           Savings of type h households
 YF              Business income
 YH(h)           Income of type h households
* Other variable
 LEON            Excess supply on the market for services
;
```

Declaration of equations

As with the previous components of the model, 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 XSEQ(j) is followed by a short description. Use a semicolon to indicate the end once all equations have been declared.

Assignment of equations

The syntax for defining equations is as follows. First, the name and dimension is stated followed by two dots. Then the equation is defined, ending 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 represent "greater than" and "less than" respectively. 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 as all equations are solved simultaneously.

In the example below, we have grouped equations by category using multiple **EQUATIONS** statements. Each time the **EQUATIONS** statement is used, it ends with a semi-colon.

```
* Production
EQUATIONS
 XSEQ(j)          Value added demand in industry j (Leontief)
 CIEQ(j)          Total intermediate consumption demand in industry j (Leontief)
 VAEQ(j)          Cobb-Douglas between labour and capital
 LDEQ(j)          Demand for labour by industry j
 KDEQ(j)          Demand for capital by industry j
 DIEQ(i,j)        Intermediate consumption of commodity i by sector j
;

 XSEQ(j)..        VA(j) =e= v(j)*XS(j);
 CIEQ(j)..        CI(j) =e= io(j)*XS(j);
 VAEQ(j)..        VA(j) =e= A(j)*LD(j)**alpha(j)*KD(j)**(1-alpha(j)) ;
 LDEQ(j)..        W*LD(j) =e= alpha(j)*PVA(j)*VA(j);
 KDEQ(j)..        R(j)*KD(j) =e= (1-alpha(j))*PVA(j)*VA(j);
 DIEQ(i,j)..      DI(i,j) =e= aij(i,j)*CI(j);

* Income and savings
EQUATIONS
 YHSEQ            Household income (workers)
 YHCEQ            Household income (capitalists)
 SHEQ(h)          Household h savings
 CTHEQ(h)         Consumption budget
 YFEQ             Firms income
 SFEQ             Firms savings
;

 YHSEQ..          YH('sal') =e= W*SUM[j,LD(j)];
 YHCEQ..          YH('cap') =e= lambda*SUM[j,R(j)*KD(j)]+DIV;
 SHEQ(h)..        SH(h) =e= psi(h)*YH(h);
 CTHEQ(h)..       CTH(h) =e= YH(h)-SH(h);
 YFEQ..           YF =e= (1-lambda)*SUM[j,R(j)*KD(j)];
 SFEQ..           SF =e= YF-DIV;
```

```
* Demand
EQUATIONS
 CEQ(i,h)         Household h consumption of commodity i
 INVEQ(i)         Investment in commodity i
 DITEQ(i)         Intermediate demand for commodity i
;

 CEQ(i,h)..       P(i)*C(i,h) =e= gamma(i,h)*CTH(h);
 INVEQ(i)..       P(i)*INV(i) =e= mu(i)*IT;
 DITEQ(i)..       DIT(i) =e= SUM[j,DI(i,j)];

* Prices
EQUATIONS
 PCIEQ(j)         Intermediate consumption price index
 CPEQ(j)          Production costs for sector j
;

 PCIEQ(j)..       PCI(j)*CI(j) =e= SUM[i,P(i)*DI(i,j)];
 CPEQ(j)..        P(j)*XS(j) =e= PVA(j)*VA(j)+PCI(j)*CI(j);

* Equilibrium
EQUATIONS
 PEQ(bns)         Domestic absorption
 WEQ              Labour market equilibrium
 REQ(j)           Capital market equilibrium
 ITEQ             Investment-savings equilibrium
;
 PEQ(bns)..       XS(bns) =e= SUM[h,C(bns,h)]+DIT(bns)+INV(bns);
 WEQ..            LS =e= SUM[j,LD(j)];
 REQ(j)..         KS(j) =e= KD(j);
 ITEQ..           IT =e= SUM[h,SH(h)]+SF;

* Other
EQUATIONS
 WALRAS           Verification of the Walras law
;

WALRAS..          LEON =e= XS('ser')-SUM(h,C('ser',h))-DIT('ser')-INV('ser');
```

Initialization procedure

A value must now be assigned to all variables previously declared. In order to solve the equations, GAMS needs to start from a benchmark value. If no value is associated with a variable, GAMS will use zero, which could cause problems if the variable appears as the denominator of a division. To do so, the suffix .L (for level) is used as follows:

```
* Initialisation of variables

C.l(i,h)          = CO(i,h);
CI.l(j)           = CIO(j);
CTH.l(h)          = CTHO(h);
DI.l(i,j)         = DIO(i,j);
DIT.l(i)          = DITO(i);
DIV.l             = DIVO;
INV.l(i)          = INVO(i);
IT.l              = ITO;
KD.l(j)           = KDO(j);
KS.l(j)           = KSO(j);
LD.l(j)           = LDO(j);
LS.l              = LSO;
P.l(i)            = PO(i);
PCI.l(j)          = PCIO(j);
PVA.l(j)          = PVAO(j);
R.l(j)            = RO(j);
SF.l              = SFO;
SH.l(h)           = SHO(h);
VA.l(j)           = VAO(j);
W.l               = WO;
XS.l(j)           = XSO(j);
YF.l              = YFO;
YH.l(h)           = YHO(h);
```

For the exogenous variables, the suffix **.FX** (for fixed) is used:

```
* Closures

* P(AGR) is the numéraire
P.fx('agr')       = PO('agr');

* Capital is sector specific
KS.fx(j)          = KSO(j);

* Total labour suply is fixed
LS.fx             = LSO;

* Dividends are exogenous
DIV.fx            = DIVO;
```

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

Executing the model

Finally, two lines must be added to solve the equation system. The first defines the model. It begins with the **MODEL** statement, followed by a name (AUTA in the example), a brief description (AUTARKY WITHOUT GOVERNMENT), the list of equations to be resolved between slashes, and end s with a semicolon. When all equations are to be solved, use the keyword **ALL** instead of the complete list of equations.

The second line defines the procedure to be used to resolve the model. The **SOLVE** statement is followed by the name of the model, **USING** and the procedure to be used to resolve the model. The procedure to be used is determined by the type of model to be solved. 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 as some are efficient for resolving certain systems but less efficient in other cases. In our example, the **CNS** (Constrained non-linear system) procedure is used.

```
MODEL AUTA Autarky without government /ALL/;
SOLVE AUTA USING CNS;
```

Reading a GAMS output file

Once GAMS has solved an equation system, it creates an output file. This section describes the output file of the program presented in the previous section.

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 reference in the file. If there are errors in the program, this is the only part of the output that will appear. For 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.

```
136   alpha(i)         = WO*LDO(i)/{PVAO(i)*VAO(i);
****                                            $8
```

Because of this error, GAMS was unable to solve the model, as such, another error is indicated on the last line of the program.

```
306  SOLVE AUTA USING CNS;
****                      $257
```

A description of the error code is given at the end of the program's echo print to facilitate correction:

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

When no errors are encountered, the following information also appears in the listing.

Displayed parameters

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

```
----    153 PARAMETER A  Scale parameter (Cobb-Douglas - production function)

AGR 1.755,    MAN 1.960,    SER 1.890


----    153 PARAMETER alpha  Elasticity (Cobb-Douglas - production function)

AGR 0.750,    MAN 0.400,    SER 0.667
```

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 written between parenthesis (LHS= " "). When the value of the left hand side differs from the one on the right hand side, GAMS add four asterisks (****). This can be useful for finding problems linked to incorrect parameter calibration or improper initialization of the variable.

```
---- XSEQ  =E=  Value added demand in industry j (Leontief)

XSEQ(AGR)..  VA(AGR) - 0.8*XS(AGR) =E= 0 ; (LHS = 0)

XSEQ(MAN)..  VA(MAN) - 0.4*XS(MAN) =E= 0 ; (LHS = 0)

XSEQ(SER)..  VA(SER) - 0.5*XS(SER) =E= 0 ; (LHS = 0)
```

Column listing

Following the equation listing, GAMS prints the equivalent information by column instead of by 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, consumption (noted C) has no lower bounds (thus -INF for minus infinity), no upper bounds (+INF) and the levels of initialization are 162, 21 and 108. The variable appears in two equations, CEQ(I,H) and PEQ(I).

```
---- C  Consumption of commodity i by type h households

C(AGR,SAL)
                  (.LO, .L, .UP, .M = -INF, 162, +INF, 0)
         1        CEQ(AGR,SAL)
        -1        PEQ(AGR)

C(AGR,CAP)
                  (.LO, .L, .UP, .M = -INF, 21, +INF, 0)
         1        CEQ(AGR,CAP)
        -1        PEQ(AGR)

C(MAN,SAL)
                  (.LO, .L, .UP, .M = -INF, 108, +INF, 0)
       (1)        CEQ(MAN,SAL)
        -1        PEQ(MAN)
```

Model statistics

The purpose of this section is to provide information on the size and non-linearity of the model. The BLOCK gives the number of equations and variables in the model, 22 and 21 in our example respectively, while SINGLE gives the number of rows and columns generated by the solver, 58 and 58 in this case. The NON-ZERO ELEMENTS count refers to the number of non-zero coefficients. The four remaining entries provide information about the non-linearity of the model. Finally, this section indicates the time used to generate the model (GENERATION TIME).

```
MODEL STATISTICS

BLOCKS OF EQUATIONS          22     SINGLE EQUATIONS           58
BLOCKS OF VARIABLES          21     SINGLE VARIABLES           58
NON ZERO ELEMENTS           193     NON LINEAR N-Z             92
DERIVATIVE POOL              20     CONSTANT POOL              24
CODE LENGTH                 195
FIXED EQUATIONS              58     FREE VARIABLES             58


GENERATION TIME      =  0.000E+0000 SECONDS      4 MB  24.6.1 r55820 WEX-WEI
```

Solve Summary

While the previous sections explained the solving process, this section concerns the solution itself. The first topic on this matter is the solve summary. It recalls the name of the model (here AUTA), the type of model (CNS) and the solver used (CONOPT). 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 "16 Solved" indicates a solution was found for the equation system.

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

     MODEL   AUTA
     TYPE    CNS
     SOLVER  CONOPT               FROM LINE  334

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      16 Solved
```

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 and the level. While the bounds are defined by the modeller, the level is calculated by the solver. A dot "." means zero. Here is an example of the equation list:

```
---- EQU XSEQ  Value added demand in industry j (Leontief)

       LOWER     LEVEL     UPPER

AGR      .         .         .
MAN      .         .         .
SER      .         .         .
```

And an example of the variable listing:

```
---- VAR C  Consumption of commodity i by type h households

            LOWER     LEVEL     UPPER

AGR.SAL      -INF    162.000     +INF
AGR.CAP      -INF     21.000     +INF
MAN.SAL      -INF    108.000     +INF
MAN.CAP      -INF     84.000     +INF
SER.SAL      -INF    270.000     +INF
SER.CAP      -INF    105.000     +INF
```