

# Introduction to GAMS

## 1 GAMS Setup and Usage

The HPC Linux computers in ETC 3.140 all have GAMS residing in the directory “/usr/local/gams/”. This should be in your path, but if it is not you can either (i) append “/usr/local/gams:” to your `.profile` file and subsequently type “gams filename” (logging off and then on again will ensure the path is included) or (ii) type “/usr/local/gams/gams filename”. If you want to see what components are installed, you may take a look at the install log file: `/local/gams/gamsinst.log`

CPLEX is also installed, and is under the path `/local/gams/cplex/`. You may use a web browser at any HPC computer to look at the manual ”`/local/gams/cplex/cplexman.html`”

To use the student PC GAMS version (which has an upper bound of 1000 nonzeros) follow the installation directions that came with your software and (I’d recommend you) select MINOS5 as the default LP solver. You’ll also need to include the `gams` path in the `autoexec.bat` file.

In general, to run GAMS type: `gams filename option1 option2 ...` GAMS looks for a file called `filename` and if it does not find such a file then looks for a file called `filename.gms`; i.e., it is not necessary to specify the `.gms` extension. It is not necessary to include any options: `gams filename` is called the “no frills” option. Simply typing `gams` on the PC or `gams -h` on the RS6000 will give you more details on the available options (the RISC machine command also contains a number of UNIX-specific remarks). In addition, on the PC you can examine the file `gamsparm.doc` for yet more details on the options. Default values for these options are specified in the file `gamsparm.txt`; these values may be overwritten on the command line.

After having typed (some variant of) `gams filename`, model generation and algorithm progress information should appear on the screen, but the bulk of the output (under the default) will go to a file called `filename.lst`. The output can be re-directed via a command option.

GAMS comes with a number of example formulations; these are listed in the user’s guide and contained in the `modlib` subdirectory within the `gams` directory. To retrieve a `.gms` file simply type `gamslib filename`. For example, `gamslib trnsport` retrieves the `trnsport.gms` file discussed in the second chapter (tutorial) of the GAMS manual.

## 2 GAMS on the Internet

WWW homepage: <http://www.gams.com/>

There is a GAMS listserv: To subscribe send an e-mail message to `listserv@vm.gmd.de` with the body of the message `SUBSCRIBE GAMS-L first-name last-name`

You will subsequently receive information regarding the GAMS-L listserv.

### 3 An Example: BJ&S problem 1.1 – A Feed-Mix Model

#### 3.1 Mathematical Formulation

**Indices/Sets:**

$i \in I$  ingredients: corn, limestone, soybeans, fish-meal

$j \in J$  products: cattle-feed, sheep-feed, chicken-feed

$k \in K$  nutrients: vitamins, protein, calcium, crude-fat

**Data:**

$d_j$  demand for product  $j$  (mtons)

$s_i$  supply of ingredient  $i$  (mtons)

$l_{jk}$  lower bound on number of units of nutrient  $k$  (per mton) for each product  $j$  (raw data  $\times$  1k)

$u_{jk}$  upper bound on number of units of nutrient  $k$  (per mton) for each product  $j$  (raw data  $\times$  1k)

$c_i$  cost per mton of ingredient  $i$  (raw data  $\times$  1k)

$a_{ik}$  units of nutrient  $k$  per mton of ingredient  $i$  (raw data  $\times$  1k)

**Decision Variables:**

$x_{ij}$  amount (in mtons) of ingredient  $i$  used for making product  $j$

**Formulation:**

$$\text{minimize } z = \sum_{i \in I} \sum_{j \in J} c_i x_{ij}$$

$$\text{subject to } \sum_{i \in I} x_{ij} = d_j \quad \forall j \in J$$

$$\sum_{j \in J} x_{ij} \leq s_i \quad \forall i \in I$$

$$\sum_{i \in I} a_{ik} x_{ij} \leq u_{jk} d_j \quad \forall j \in J, k \in K$$

$$\sum_{i \in I} a_{ik} x_{ij} \geq l_{jk} d_j \quad \forall j \in J, k \in K$$

$$x_{ij} \geq 0 \quad \forall i \in I, j \in J$$

## 3.2 GAMS Formulation

Every GAMS formulation follows the basic format suggested by the above formulation (lifted straight out of chapter two of the GAMS manual):

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

There is actually considerable flexibility with respect to the ordering possibilities of the above structure; see the discussion in chapter 3 of the manual. The important rule to remember is that something must be declared (but not necessarily assigned or defined) prior to use.

The following is a sample file named `feedmix.gms`. To run this file with the no frills gams call you'd type `gams feedmix` and the output would go to a file named `feedmix.lst`.

```
$TITLE BJ&S problem 1.1 -- a feed-mix blending model

SETS
    I ingredients /CORN, LIMESTONE, SOYBEANS, FISH-MEAL/
    J products    /CATTLE-FD, SHEEP-FD, CHICKEN-FD/
    K nutrients   /VITAMINS, PROTEIN, CALCIUM, CRUDE-FAT/;

PARAMETERS
    DEMAND(J) demand for product j (in metric tons)
    / CATTLE-FD    10
      SHEEP-FD     6
      CHICKEN-FD   8 /

    SUPPLY(I) supply of each ingredient i (in metric tons)
    / CORN         6
      LIMESTONE    10
      SOYBEANS     4
      FISH-MEAL    6 /;

TABLE LOWER(J,K) lower bound on number of units of nutrient k (per
*                kilogram) for each product j
                VITAMINS PROTEIN CALCIUM CRUDE-FAT
CATTLE-FD      6.0    6.0    7.0    4.0
SHEEP-FD       6.0    6.0    6.0    4.0
CHICKEN-FD     4.0    6.0    6.0    4.0 ;
```

TABLE UPPER(J,K) upper bound on number of units of nutrient k (per  
 \* kilogram) for each product j  
 VITAMINS PROTEIN CALCIUM CRUDE-FAT  
 CATTLE-FD INF INF INF 8.0  
 SHEEP-FD INF INF INF 6.0  
 CHICKEN-FD 6.0 INF INF 6.0 ;

PARAMETER COST(I) cost per kilogram of ingredient i  
 / CORN 0.20  
 LIMESTONE 0.12  
 SOYBEANS 0.24  
 FISH-MEAL 0.12 /;

TABLE ATTRIBUTE(I,K) # of units of nutrient k per kilogram of ingredient i  
 VITAMINS PROTEIN CALCIUM CRUDE-FAT  
 CORN 8.0 10.0 6.0 8.0  
 LIMESTONE 6.0 5.0 10.0 6.0  
 SOYBEANS 10.0 12.0 6.0 6.0  
 FISH-MEAL 4.0 8.0 6.0 9.0 ;

SCALAR KGPTON kilograms per ton /1000.0/;

LOWER(J,K)=LOWER(J,K)\*KGPTON;  
 UPPER(J,K)=UPPER(J,K)\*KGPTON;  
 COST(I)=COST(I)\*KGPTON;  
 ATTRIBUTE(I,K)=ATTRIBUTE(I,K)\*KGPTON;

VARIABLES

X(I,J) amount of ingredient (in metric tons) of ingredient i  
 \* used for making product j  
 TOTALCOST total cost of the solution ;

POSITIVE VARIABLE X;

EQUATIONS

OBJ defines the objective function - minimize total cost  
 DEMANDCON(J) meet demand for product j  
 SUPPLYCON(I) observe supply for ingredient i  
 NUTRIENTLO(J,K) satisfy minimum nutrient requirement: feed j nutrient k  
 NUTRIENTUP(J,K) satisfy maximum nutrient requirement: feed j nutrient k;

OBJ.. TOTALCOST =E= SUM( I,J, COST(I)\*X(I,J) ) ;

DEMANDCON(J).. SUM( I, X(I,J) ) =E= DEMAND(J) ;

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

NUTRIENTLO(J,K).. SUM( I, ATTRIBUTE(I,K)\*X(I,J) ) =G=  
 LOWER(J,K)\*DEMAND(J) ;

NUTRIENTUP(J,K).. SUM( I, ATTRIBUTE(I,K)\*X(I,J) ) =L=  
 UPPER(J,K)\*DEMAND(J);

MODEL FEEDMIX /ALL/ ;

SOLVE FEEDMIX USING LP MINIMIZING TOTALCOST;

### 3.3 Notes on the GAMS Formulation

0. Every GAMS statement must be terminated with a semicolon ;
1. \$TITLE is an example of a *dollar control directive*. Every page of the listing (.lst) file will echo the (up to 80) characters that follow \$TITLE. If the \$TITLE line is omitted GAMS will simply substitute its own title at the heading of each page.
2. The GAMS compiler is not case sensitive; a common convention is to use uppercase letters for GAMS code and lowercase text for *documentation*. Any line that has an asterisk, \*, in the first column is a comment line and will be ignored by the GAMS compiler. It is also possible to insert comments within certain GAMS statements; the lower case text in the feed-mix program provide examples of this type of documentation. A block of (typically multiline) text that is preceded by \$ONTEXT and followed by \$OFFTEXT (again with the \$ in the first column) is ignored. In-line comments, as in Pascal, are permitted provided the dollar control directive \$INLINECOM { } is included.
3. The following provide two equivalent ways of declaring the three sets used in this model:

```
SETS I ingredients
      /
      CORN      grows on ears
      LIMESTONE found in caves
      SOYBEANS  found in hotdogs
      FISH-MEAL don't ask
      /
J products /CATTLE-FD cows eat it, SHEEP-FD sheep eat it,
           CHICKEN-FD you guessed it/
K nutrients /VITAMINS, PROTEIN, CALCIUM, CRUDE-FAT/;

SET I ingredients /CORN, LIMESTONE, SOYBEANS, FISH-MEAL/ ;
SET J products   /CATTLE-FD, SHEEP-FD, CHICKEN-FD/ ;
SET K nutrients  /VITAMINS, PROTEIN, CALCIUM, CRUDE-FAT/;
```

The word SET and SETS are equivalent. The text `ingredient`, `products`, and `nutrients` could be eliminated with no difference from the compiler's perspective. The asterisk can be useful for declaring sets with elements that involve numerical sequences.

```
SETS I / I1 * I10 /
      J / J01C * J05C /;
```

is equivalent to

```
SETS I / I1, I2, I3, I4, I5, I6, I7, I8, I9, I10 /
      J / J01C, J02C, J03C, J04C, J05C /;
```

More on SETS: ALIAS, subsets, and multidimensional sets are discussed in Chapter 4 of the manual.

4. GAMS calls the elements of a set *labels*. Labels can contain at most 10 characters so that CHICKEN-FEED would be illegal. A label must start with a letter and can contain letters, numbers, and + and - but no other special characters (more details in §3.4 of the manual.) Other identifiers (such as names given to SETS, PARAMETERS, VARIABLES, MODELS, etc.) follow slightly more stringent rules in that no special characters (e.g., + and -) are permitted.

5. Data is entered via `PARAMETERS`. The GAMS compiler regards `SCALARS` and `TABLES` as identical to `PARAMETERS`; the difference from the user's perspective lies only in methods for declaration and initialization with numerical values.

6. GAMS has a feature called *domain checking*. For example, the domain of `DEMAND` is the set `J`; if the assignment of values to `DEMAND` uses a label that is not an element of `J` a domain violation error will be reported on compilation. The compiler does not care about the order of the entries in the `DEMAND` assignment; in other words, they need not be the same order as the assignment in `J`. If a parameter value is not assigned then it will take on value zero; for example, if the `SHEEP-FD` 6 entry was not included in the `DEMAND` assignment then `DEMAND("SHEEP-FD")` takes on the value zero. An equivalent set of GAMS code for declaring and assigning values to `DEMAND` is:

```
PARAMETER DEMAND(J) demand for product j (in metric tons);
DEMAND("CATTLE-FD")=10.0;
DEMAND("SHEEP-FD")=6.0;
DEMAND("CHICKEN-FD")=8.0;
```

7. `TABLES` provide a convenient way to enter two (or higher) dimensional `PARAMETERS`. As you may have already guessed from the remark above, an equivalent way to enter `LOWER` would be

```
PARAMETER LOWER(J,K) lower bound on ...;
LOWER("CATTLE-FD","VITAMINS")=6.0;
LOWER("CATTLE-FD","PROTEIN")=6.0;
LOWER("CATTLE-FD","CALCIUM")=7.0;
...
LOWER("CHICKEN-FD","CRUDE-FAT")=4.0;
```

There is a less tedious alternative to this option

```
PARAMETER LOWER(J,K) lower bound on ...
/
CATTLE-FD.VITAMINS  6.0
CATTLE-FD.PROTEIN   6.0
CATTLE-FD.CALCIUM   7.0
...
CHICKEN-FD.CRUDE-FAT 4.0
/
;
```

8. We can extend the ideas from remark 7; the `TABLE` method of defining `UPPER` can be replaced by the following code.

```
PARAMETER UPPER(J,K)
/
CATTLE-FD.CRUDE-FAT           = 8.0
SHEEP-FD.CRUDE-FAT           = 6.0
CHICKEN-FD.(VITAMINS,CRUDE-FAT) = 6.0
/;
UPPER(J,K)$(UPPER(J,K) EQ 0) = INF;
```

First, the `=` sign can be used as a valid separator of labels and values; the code would still be valid if the `=` sign were omitted (provided a space separator was present). Second, the

```
CHICKEN-FD.(VITAMINS,CRUDE-FAT)
```

trick can be used when making multiple assignments to the same value; this line is equivalent to

```
UPPER("CHICKEN-FD","VITAMINS")=6.0;  
UPPER("CHICKEN-FD","CRUDE-FAT")=6.0;
```

This trick can also be used with the asterisk; see remark 3 above and §5.1 of the manual. GAMS uses extended-range arithmetic; INF represents infinity. Other valid symbols include -INF (negative infinity), UNDF (undefined), EPS (epsilon), and NA (not available). The \$-operator is used to make *conditional* assignments or definitions. If \$(UPPER(J,K) EQ 0) were absent in the above code then UPPER(J,K) would be assigned INF for all (J,K) pairs – overwriting the previous four assignments. However, in its existing form the assignment is made only for those UPPER's that are currently zero (recall that uninitialized parameters take on value zero). We introduce two more constructs IF-ELSE and LOOP to accomplish the same thing (in a more complicated and less efficient way).

```
PARAMETER UPPER(J,K)  
/  
CATTLE-FD.CRUDE-FAT           = 8.0  
SHEEP-FD.CRUDE-FAT           = 6.0  
CHICKEN-FD.(VITAMINS,CRUDE-FAT) = 6.0  
/  
LOOP((J,K),  
    IF(UPPER(J,K) EQ 0,  
        UPPER(J,K) = INF;  
    )  
);
```

9. SCALAR provides a method for introducing a zero dimensional PARAMETER; i.e., one that has exactly one numerical value.

10. PARAMETERS may be manipulated once they have been introduced. Examples in the feed-mix GAMS program include the rescaling of certain data to metric tons.

```
LOWER(J,K)=LOWER(J,K)*KGPTON;  
UPPER(J,K)=UPPER(J,K)*KGPTON;  
COST(I)=COST(I)*KGPTON;  
ATTRIBUTE(I,K)=ATTRIBUTE(I,K)*KGPTON;
```

The statement LOWER(J,K)=LOWER(J,K)\*KGPTON has an implicit loop; the assignment is made for all elements of the Cartesian product of J and K. Note that we could reduce the computational effort of the

```
UPPER(J,K)=UPPER(J,K)*KGPTON;
```

statement by using the \$-operator.

```
UPPER(J,K)$ (UPPER(J,K) LT INF)=UPPER(J,K)*KGPTON;
```

In this case, this trick amounts to minimal (if any) savings, but if the right-hand-side contained more complicated and expensive operations the savings could be substantial.

11. The DISPLAY statement can be used to display the values of parameters, variables, etc. For example, suppose you wanted to examine the values of UPPER before and after the multiplication by KGPTON. To do this, you could insert

```

DISPLAY "prior to multiplication by KGPTON", UPPER;
...
DISPLAY "after multiplication by KGPTON", UPPER;

```

in the appropriate places. The corresponding output from the `feedmix.lst` file is displayed below

```

----      62 prior to multiplication by KGPTON
----      62 PARAMETER UPPER
           VITAMINS      PROTEIN      CALCIUM      CRUDE-FAT

CATTLE-FD      +INF      +INF      +INF      8.000
SHEEP-FD       +INF      +INF      +INF      6.000
CHICKEN-FD     6.000      +INF      +INF      6.000

----      84 after multiplication by KGPTON
----      84 PARAMETER UPPER
           VITAMINS      PROTEIN      CALCIUM      CRUDE-FAT

CATTLE-FD      +INF      +INF      +INF     8000.000
SHEEP-FD       +INF      +INF      +INF     6000.000
CHICKEN-FD     6000.000      +INF      +INF     6000.000

```

**12.** The `ABORT` statement provides a method for verifying data integrity (it has other uses as well). In the feed-mix problem the lower bound on the number of nutrients for any product-nutrient pair should be smaller than the corresponding upper bound. The following code provides a method for testing this.

```

SCALAR LUTEST;
LUTEST=SMIN((J,K),UPPER(J,K)-LOWER(J,K));
ABORT$(LUTEST LT 0) "LOWER exceeds UPPER", LUTEST

```

If the condition in the `ABORT` statement is true GAMS will report an “execution” error and the output “LOWER exceeds UPPER”, `LUTEST` will be in the `.lst` file. `SMIN` is a set-minimum function; `MIN` should be used if the argument list contains scalars.

**13.** `VARIABLES` may be declared one of five types: `FREE`, `POSITIVE`, `NEGATIVE`, `BINARY`, and `INTEGER`. `FREE` is the default type and such a variable is unrestricted in sign. The rest of the types are self-explanatory with the exception of the `INTEGER` variable that can take on values 0, 1, 2, ..., 99, 100. *Note:* If you have an integer program in which 100 is not sufficiently large, use the `.UP` suffix to adjust the upper limit (see 14 below).

```

POSITIVE VARIABLES
  X(I,J)      amount of ingredient (in metric tons) of ingredient i
*            used for making product j;
FREE VARIABLE TOTALCOST      total cost of the solution ;

```

represents a valid alternative to the existing code.

**14.** A `PARAMETER` has one numerical value associated with each label but a `VARIABLE` has four numerical values; these are referenced by suffices: `.LO` (lower bound), `.UP` (upper bound), `.L` (activity level), and `.M` (marginal or dual value, or reduced cost). Lower and upper bounds are specified by the user; for example, by declaring `X` to be a `POSITIVE` (actually non-negative) variable we have effectively set `X.LO(I,J)=0.0` and `X.UP(I,J)=INF`. In the current model, it is clear from the `DEMAND` values that we can include the upper bounds

```
X.UP(I,"CATTLE-FD")=10.0;
X.UP(I,"SHEEP-FD")=6.0;
X.UP(I,"CHICKEN-FD")=8.0;
```

without altering the optimal solution. Sometimes it is convenient to introduce a “decision” variable whose value is fixed; i.e., its upper and lower bound take on the same value. This can be accomplished using the `.FX` suffix; this is not really a new suffix (from the compiler’s perspective) because it is just shorthand for making two statements involving `.LO` and `.UP`. The statement `DISPLAY X;` will yield an error indicating that a suffix must be specified; e.g., `DISPLAY X.L;`.

**15.** The term `EQUATIONS` should be interpreted liberally as it represents all types of structural constraints; i.e., inequality constraints in addition to equality (equation) constraints. One twist in GAMS is that the objective function value (in our case `TOTALCOST`) must be defined as a (scalar) variable and the objective function defined as a (scalar) accounting constraint (in our case with the identifier `OBJ`). The `SUM` operator in the `OBJ` constraint (from the feed-mix model) sums across the Cartesian product of sets `I` and `J`. An equivalent statement of this double sum is:

```
OBJ.. TOTALCOST =E= SUM(I,
                        COST(I)*SUM(J,X(I,J))
                      );
```

**16.** The `$`-operator can be used to control the subset over which a sum occurs; it can also be used to control the subset over which a set of equations is defined. The size of the feed-mix formulation can be reduced by using the latter trick. In particular, there is no need to generate the `NUTRIENTUP` constraints for which the corresponding product-nutrient pair that have an infinite value of `UPPER`. We can exclude these constraints from the model by using the following code:

```
NUTRIENTUP(J,K)$ (UPPER(J,K) LT INF) ..
                SUM( I, ATTRIBUTE(I,K)*X(I,J) ) =L= UPPER(J,K)*DEMAND(J);
```

This reduces the size of the model from  $32 \times 13$  (133) to  $14 \times 13$  (101) where this notation means # of rows  $\times$  # of columns (# of nonzeros).

**17.** The `MODEL` statement gives the model a name and determines which constraints are included in the problem formulation. The general usage also permits comment-type text between the *model name* (in our case `FEEDMIX`) and the *equation list* (in our case we’ve included all declared equations via `ALL`). Models require a name because multiple models can be declared within the same program. For example, we could declare two models: one with all the constraints and one with no supply constraint

```
MODEL FEEDMIX1 the original model /ALL/;
MODEL FEEDMIX2 supply constraint dropped /OBJ, DEMANDCON, NUTRIENTLO,
NUTRIENTUP/;
```

**18.** A GAMS formulation can also contain multiple `SOLVE` statements. To continue the example above, we could solve the original and relaxed models via

```
SOLVE FEEDMIX1 USING LP MINIMIZING TOTALCOST;
SOLVE FEEDMIX2 USING LP MINIMIZING TOTALCOST;
```

**19.** The feed-mix problem (as specified in BJ&S) is infeasible; this can be determined by running the GAMS model shown above. A careful examination of the crude-fat upper bounds (in the context of the rest of the model) indicates why this is the case. Infeasibility can be detected by searching for INFES in the listing file; this will indicate which constraints are infeasible in the “solution” that the solver found. Sometimes this is helpful but in large models it sometimes provides little insight (the real cause may be masked). One way to convert the feed-mix model into a feasible problem is increase the UPPER values associate with CRUDE-FAT:

```
UPPER(J, "CRUDE-FAT")=8.0;  
UPPER(J, "CRUDE-FAT")=UPPER(J, "CRUDE-FAT")*KGPTON;  
SOLVE FEEDMIX USING LP MINIMIZING TOTALCOST;
```

**20.** The default listing file contains:

- (1) An echo print of the GAMS program. \$OFFUPPER will ensure this echo is in the same mixed case in which it was entered. If, for some reason, you want the echo in all upper case use \$ONUPPER.
- (2) Two reference maps. These can be useful for debugging; \$OFFSYMREF OFFSYMLIST will suppress these two maps. Two additional maps can be turned on via \$ONUPELLIST ONUELXREF.
- (3) An equation listing. These can be *extremely* useful for debugging. OPTION LIMROW=3 is the default and it means (up to) three equations will be listed for each equation block. 3 can be replace with any non-negative integer; 0 means that no equations will be listed.
- (4) Column listing. This is analagous to the equation listing but for variables. It lists the equations in which the variable appears as well as .LO, .L, and .UP values. OPTION LIMCOL=3 is the default.
- (5) Model Statistics. Brief summary of model size.
- (6) Solve Summary. Brief summary of after the solver terminated. MODEL STATUS for instance will indicate whether the model is infeasible or whether an optimal solution was found, etc. SOLVER STATUS, on the other hand, gives some information as to how the solver terminated.
- (7) Solution Listing. Row-by-row and column-by-column listing of the solution. The listing can be suppressed by OPTION SOLPRINT=OFF; you may wish to do this and instead use DISPLAY to examine selected parts of the solution.

**21.** Sometimes lines that fit in the .gms file will be too wide in the .lst file because line numbers are put in front; moreover, a wider page may be useful for displaying certain solutions. The PW command line option can be used to control the page width; the minimum value is 72 and the maximum is usually 255 (but can be machine dependent). Similarly, pagebreaks in the listing file can be undesirable if output needs to be exported to another program (it can also be annoying from an on-screen readability perspective). The PS command line option can be used to control the page size; the minimum value is 30 and the maximum value is 9999.

```
gams feedmix pw=120 ps=9999
```

will produce a listing file that is 120 characters wide and has no pagebreaks (actually pagebreaks will still exist between certain sections of the listing file).

**22.** Premature termination of the solver may occur due to the default values for ITERLIM (iteration limit; default is 1000) and R ESLIM (resource limit; default is 1000). The units of RESLIM are machine

dependent (typically some mixture of CPU or Wall Clock and seconds or minutes. `OPTION LP=MINOS5` is used to select the MINOS 5.0 solver. Use `OPTION SUBSYSTEMS` to determine what solvers are available (and how to refer to them).

**23.** `$INCLUDE "filename"` (quotes optional) can be used (multiple times) to include a file; this is desirable when there is a large amount of data and/or data is generated from another program. You can also have nested includes (to a limited extent) and the include file could be another GAMS program (although there is currently no such thing as “local” declarations).

**24.** `ORD` and `CARD` are very useful functions, particularly when dealing with a time- (or otherwise) dynamic model. For example consider

```
SET T /PERIOD1*PERIOD20/
```

`CARD(T)` denotes the cardinality of `T`; in this case it has value 20. Mathematical sets contain no ordering, but GAMS recognizes order in certain types of sets. Such sets must be one-dimensional, static, and *ordered*; see §12.1 of the manual for more on ordering (in most cases where you’d want to use it you have an ordered set). Thus `ORD("PERIOD1")` has value 1 and `ORD("PERIOD2")` has value 2, etc.

**25.** Lead and lag operators. Consider

```
SETS T /T1*T10/;
PARAMETER X(T);
X(T)=ORD(T);
X(T)=X(T)+X(T+1);
DISPLAY X;
```

The output of the display statement is

```
----          7 PARAMETER X
T1   3.000,    T2   5.000,    T3   7.000,    T4   9.000,    T5  11.000
T6  13.000,    T7  15.000,    T8  17.000,    T9  19.000,    T10 10.000
```

The expression `T+1` is clearly not standard algebra since `T` is a set and its elements (labels) do not have numerical values. `X("T1"+1)` is simply `X("T2")`; the `+` operator advances to the next element of the set. This only makes sense for ordered sets. It is also possible to use the `-` operator in the same context. This should make all the above values clear, with the possible exception of `X("T10")` which was assigned `X("T10")=10+X("T10"+1)`. GAMS gives `X("T10"+1)`, which does not exist, the value zero. The operators `++` and `--` provide for “circular” lead and lag operators; for example `"T1"--3` would represent `"T8"`. See §12.3 for more information.

26. “\$ on the left” and “\$ on the right.” Consider two separate code segments. First,

```
set i /i1*i5/;
parameter a(i);
a(i)=37;
a(i)$ (ord(i) le 2) = ord(i);
display a;
```

and second,

```
set i /i1*i5/;
parameter a(i);
a(i)=37;
a(i) = ord(i)$ (ord(i) le 2);
display a;
```

The first code segment has the “\$ on the left” of the =, and the second code segment has the “\$ on the right” of the =.

The output of the first code segment is

```
----          PARAMETER A
I1 1.000,  I2 2.000,  I3 37.000,  I4 37.000,  I5 37.000
```

and the output of the second code segment is

```
----          PARAMETER A
I1 1.000,  I2 2.000
```

a(i3)=a(i4)=a(i5)=0 for the second code segment.

27. See Chapter 13 of the manual for information on how to control the format of DISPLAY statements.