GAMS features

Maria Blanco

Dep. Agricultural Economics
Technical University of Madrid
maria.blanco@upm.es
Overview

- Sets and mappings
- Conditionals
- Initial values and bounds
- Display options
- Comments
- GDX utilities
- GAMS functions
- Control variables
- Conditional compilation
- Model attributes and options
- Partial run
Sets and mappings
Working with SETS – subsets

Subsets contain part of the elements of another set

- All elements of the subset must be elements of the larger set.
- The elements of the subset may be defined explicitly or may be calculated.

**Example**

```gams
set crops all crops /wheat, maize, tomato, potato, sunflower, soya, sugarbeet/
set cereals(crops) crops /wheat, maize/;
```
Working with SETS – Alias

**Alias statement**

- Gives another name to a set defined previously
- Useful in market equilibrium problems to specify cross elasticities

**Syntax**

```
alias (knownSet, newSet);
```

*example*

```gams
set c commodities;
alias (c, cc);
```
Working with SETS – dynamic sets

Subset (of a static set) whose elements can change
  ▶ The keywords used to denote membership or non-membership are YES and NO

Example

```gams
set Y 'years' / 2010*2020/
cury(y) 'current year'
;
cury('2010') = 'yes';
```

- Dynamic sets cannot be used as domains
Ord and card

- **Ord**: parameter that indicates the relative position of each element in the set
- **Card**: scalar that indicates the number of elements in the set

**Example**

```gams
set Y 'years' /2011*2020/;
parameter p_ord, p_card;
p_ord(y) = ord(y);
p_card = card(y);
display p_ord, p_card;
```
Working with SETS – lead and lag

Lead and lag

➤ Lead-lag effect
➤ Links between variables over time

Example

```plaintext
set Y 'years' /2011*2020/ ;

parameter pop(y) 'population' /2011 100/
    grate(y) 'growth rate'
;
    grate(y) = 0.02;

loop(y, pop(y+1) = pop(y) *(1+grate(y)) ) ;

display pop;
```
Working with SETS – mappings

Multidimensional sets used to create tuples

- The keywords used to denote membership or non-membership are YES and NO

Syntax

```plaintext
set C  "crops" /wheat,maize,tomato/
T  "techniques" /T0*T2/
CT(C,T) "feasible combinations crop-technique"
;
CT(c,t) = yes;
CT('maize','t0') = no;
CT('tomato','t0') = no;
```
Conditionals
Conditionals (subsets)

Used to define parameters, variables or equations only for selected set elements

Example

```gams
set C 'crops' /wheat,maize,tomato,potato,sunflower,soya,sugarbeet/
cereals(c) 'cereals' /wheat,maize/
;
parameter SB(c) 'subsidy' ;
SB(cereals) = 150 ;
display SB;
```
Conditionals ($ operator)

Used in conditional assignments, expressions and equations

The condition: if \( x > 3 \), then \( y = 2 \)

can be modelled in GAMS as follows:

**Syntax**

\[
y$( x>3) = 2;
\]

"y, such as x is greater than 3, equals 2"
Conditionals ($ operator)

Conditional assignments

- **$ on the left**: no assignment is made unless the logical condition is satisfied
- **$ on the right**: an assignment is always made (the term will be zero when the condition is not satisfied)

Conditional equations

- Dollar operator within the algebra (analogous to $ on the right)
- Dollar control over the domain definition (analogous to $ on the left)

Let us look at some examples!
$ operator (example)

--- dollar on the left "p_left, such that a=a1, equals 50"

\[
p_{left}(a)$(ord(a)=1) = 50 ;
\]

--- dollar on the right "if a=a1, then p_right=50, else p_right=0"

\[
p_{right}(a) = 50$(ord(a)=1) ;
\]
Non-linear models (initial values and bounds)
Assigning initial values to variables

- They help GAMS to find the optimal solution and speed up the iteration process (default value is zero)
- They need to be entered before the solve statement

Syntax

```plaintext
var_name.L = ini_value;
solve nl pModel ... 
```
Working with non-linear models

Providing lower and upper bounds

- They speed up the iteration process
- They are useful when working with variables that are undefined if another variable becomes zero
- They need to be entered before the solve statement

Syntax

```
var_name.LO = lower_value ;
var_name.UP = upper_value ;
solve nl pModel . . .
```
Fixed variables

Variables can be endogenous or exogenous depending on the model run

Fixed variables

- It is possible to fix the value of a variable through the suffix `.fx` => equivalent to setting lower and upper bounds equal to the fixed value

Example

```gams
variable v(r,p,m,y) 'endogenous variables' ;

* fixed value for ad-valorem tariff
v.fx(r,p,"TAV",y) = 0 ;
```
Display options
GAMS features (DISPLAY)

DISPLAY statement:

- Instruction that allows us to choose which elements we want to display in the output file (.lst)
- We can display data, model results or calculations with data or results.
- When displaying parameters, we do not add the domain of definition

Syntax

```
display price, cost;
```
DISPLAY statement:

- When displaying model results we have to use the DISPLAY command after the SOLVE statement.
- Four values are associated to every variable and equation in the model. Hence, when displaying variables and equations, we need to specify which value we want to display.

Syntax:

```gams
display Z.L;

display QD.L, PD.L;
```
GAMS features (OPTION)

Option display

▶ To modify the display formatting

Syntax

```gams
option parName:decimals:rowItems:colItems;

* example
option result:1:1:1;
```
GAMS features (OPTION)

Option decimals

- Specifies the default number of decimal places to be printed by all subsequent display statements
- The default value is 3 and the range is from 0 to 8

Syntax

```
option decimals = number;
```

* example
```
option decimals = 1;
```
Comments
Three ways to include comments

1. To start a line with an asterisk (*) in the first position (single line comments). GAMS will ignore this line.

Syntax

* this line contains explanatory text

* this is a comment
but * this is not a comment
Three ways to include comments

2. To use $ontext-$offtext delimiters (multiple line comments). GAMS will ignore the text between delimiters.

Syntax

```
$ontext
this section contains explanatory text
$offtext
```

```
$ontext
this section contains explanatory text
$offtext
```
Three ways to include comments

3. To use the options $eolcom (end of line comment) or $inlinecom (inside line comment).

Syntax

$eol com #
$inlinecom {}

X = 1 ;  # this is a comment
Y = 2 ;  {this is also a comment} Z = 3 ;
GDX utilities
GDX utilities (GDX viewer)

GDX (GAMS data exchange) files:

- Files that store the values of one or more GAMS symbols (sets, parameters, variables and equations)

- Intermediary files (between GAMS language and another software package)

- Binary files that are portable between different platforms. They can be used:
  - To prepare data for a GAMS model
  - To pass results of a GAMS model into different programs

- In GAMSIDE, can be handled using the GDX-viewer
GDX utilities (GDX viewer)

Storing parameters in GDX format

- During execution of a GAMS model we can write to GDX files using the `execute_unload` command.
- If no path is specified, the gdx file will be written in the current project directory.

**Syntax**

```gams
execute_unload 'file_name.gdx' parameter_name;
execute_unload 'file_name.gdx';
```
GDX utilities (compilation time)

During compilation of a GAMS model, we can read data from a GDX file into GAMS:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GDXIN file_name</td>
<td>Specify the GDX file to be used for reading</td>
</tr>
<tr>
<td>$GDXIN</td>
<td>Close the current GDX input file</td>
</tr>
<tr>
<td>$LOAD S1 S2</td>
<td>Read GAMS symbols S1, S2</td>
</tr>
<tr>
<td>$LOAD S1=gdx1</td>
<td>Read GAMS symbol S1 with corresponding name gdx1</td>
</tr>
</tbody>
</table>
GDX utilities (compilation time)

During compilation of a GAMS model we can write to a GDX file:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GDXOUT file_name</td>
<td>Specify the GDX file to be used for writing</td>
</tr>
<tr>
<td>$GDXOUT</td>
<td>Close the current GDX output file</td>
</tr>
<tr>
<td>$UNLOAD S1 S2</td>
<td>Write GAMS symbols S1, S2</td>
</tr>
<tr>
<td>$UNLOAD S1=.gdx1</td>
<td>Write GAMS symbol S1 with corresponding name gdx1</td>
</tr>
</tbody>
</table>
GDX utilities (execution time)

During execution of a GAMS model we can read and write GDX files with the following statements:

* to read from GDX

    execute_load 'file_name.gdx' par1, par2=P2;

* to write to GDX

    execute_unload 'file_name.gdx' par3, par4=P4;
GDX utilities (GAMS-EXCEL link)

GDXXRW utility: allows reading from (and writing to) an Excel spreadsheet

**Syntax**

* Importing data from EXCEL

$CALL "GDXXRW EXE excel_file1.xls index=sheet1!A3";

$CALL "GDXXRW EXE file2.xls par=P2 rng=sheet2!A3 rdim=1 cdim=1"

* Exporting data from EXCEL

execute "GDXXRW EXE gdx_file1.gdx index=sheet1!A3";

execute "GDXXRW EXE file2.gdx par=P2 rng=sheet2!A3 rdi m=1 cdi m=1";
GDX2XLS is a tool to dump the complete content of a GDX file to an Excel spreadsheet (.xlsx or .xls file). Every identifier gets its own sheet in the Excel file.

**Syntax**

* Saving the file to GDX

```gams
execute_unload "sets_all.gdx" ;
```

* Exporting all sets to EXCEL

```gams
execute "gdx2xls sets_all.gdx" ;
```
GAMS functions
Set attributes

- Set elements have attributes that may be recovered during execution

**Syntax**

```
set name.attribute
```

where

- `setname` is the name of the set
- `attribute` is one of the following:
  - `ord`
  - `uel`
  - `pos`
  - `val`
  - `off`
  - `len`
Set attributes (example)

```
set_attributes.gms

p_1('ord', a) = a.ord;
p_1('pos', a) = a.pos;
p_1('off', a) = a.off;
p_1('uel', a) = a.uel;
p_1('val', a) = a.val;
p_1('len', a) = a.len;
```
### GAMS functions

#### Common mathematical functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>sum of set indexed expressions</td>
</tr>
<tr>
<td>prod</td>
<td>product of set indexed expressions</td>
</tr>
<tr>
<td>sqr</td>
<td>square of an expression or term</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root of an expression or term</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm</td>
</tr>
<tr>
<td>abs</td>
<td>absolute value</td>
</tr>
<tr>
<td>max, min</td>
<td>maximum or minimum of a set of expressions or terms</td>
</tr>
<tr>
<td>smax, smin</td>
<td>maximum or minimum of set indexed expressions or terms</td>
</tr>
</tbody>
</table>
# GAMS functions

## Basic statistical functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>normal(MEAN, STDDEV)</code></td>
<td>generates a random number with normal distribution with mean MEAN and standard deviation STDDEV</td>
</tr>
<tr>
<td><code>uniform(LOW, HIGH)</code></td>
<td>generates a random number between LOW and HIGH with uniform distribution</td>
</tr>
<tr>
<td><code>uniformInt(LOW, HIGH)</code></td>
<td>generates an integer random number between LOW and HIGH with uniform distribution</td>
</tr>
</tbody>
</table>
Some GAMS functions generate random numbers following a specified probability distribution.

**Syntax**

```gams
set N "number of draws" /n01*n40/ ;

parameter mean, sigma, p_normal, low, high, p_uniform ;

p_normal(n)  = normal(mean, sigma);

p_uniform(n) = uniform(low, high);
```

- Normal(mean, sigma) generates random numbers with normal distribution.
- Uniform(low, high) generates random numbers between LOW and HIGH with uniform distribution.
* **NORMAL** random number normally distributed
  *(mean, sigma)*
  
  \[
  \text{STAT}(s, \text{nor m}(0, 1)) = \text{normal}(0, 1);
  \]
  
  \[
  \text{STAT}(s, \text{nor m}(5, 2)) = \text{normal}(5, 2);
  \]

* **UNIFORM** random number with uniform distribution between \(x\) and \(y\)
  
  \[
  \text{STAT}(s, \text{un i f}(0, 1)) = \text{uniform}(0, 1);
  \]
  
  \[
  \text{STAT}(s, \text{un i f}(10, 60)) = \text{uniform}(10, 60);
  \]
Control variables
Control variables

Control variables are used for conditional compilation

**$setglobal** is used to define a global control variable (available throughout the code)

Global variables are destroyed using **$dropglobal**

**Syntax**

```
$setglobal varname varvalue
```

where **varname** is the name of the variable
**varvalue** can contain text or a number

```
$dropglobal varname
```
Control variables

$setlocal is used to define a local variable (accessible only in the code module where defined)

Local variables are destroyed using $droplocal

Syntax

$setlocal varname varvalue

where varname is the name of the variable varvalue can contain text or a number

$droplocal varname
Control variables (example)

Commonly used to articulate complex conditions

```gams
$setglobal simc IND

v.fx("%simc%","BA","TAV",y) = v.l("%simc%","BA","TAV",y) + 0.1 ;
```

`control_variables.gms`
Control variables (example)

Commonly used to articulate complex conditions

canvas
control_variables.gms

```gams
$setglobal simc swhe

parameter ygrowth(cact) 'yield growth';
ygrowth(cact) = 0.05;
ygrowth("%simc%") = 0.10;
```

Control variables for paths

- Control variables can also be used to indicate the relative path as in this example.
- Create control variables to define the paths (to data directory, results directory and scenario directory) and use the control variables throughout the code.

Example

```gams
$set global datadir ..\data
$set global resdir ..\results
$set global scendir ..\scen
```
Conditional compilation
$\textit{include}$

$\textit{include}$ inserts in an input file the content of an external file (data or GAMS statements)

**Syntax**

- When the file to be included is located in the current working directory:
  
  `$\text{INCLUDE} \ '\ \text{inc\_filename}'$

- When the file to be included is not located in the current working directory, the path has to be specified:
  
  `$\text{INCLUDE} \ '\ C:\inc\_file\_path\inc\_filename'`
$batinclude

$batinclude inserts in an input file the content of an external file and it also passes on arguments.

**Syntax**

```
$BATINCLUDE 'inc_filename' arg1 arg2 ...
```

- **file to be included** located in the current working directory
- **arguments used for substitution** (arguments are treated as character strings that are substituted by an argument number inside the included file)
if / else / elseif

If, else, elseif: logical conditions

**Syntax**

```gams
If (logical condition,  
    statements to be executed if true ;);  

If (logical condition,  
    statements executed if condition true;  
else  
    statements executed if condition not true;);  

If (logical condition,  
    statements to be executed if true ;  
Elseif logical condition,  
    statements executed if this conditional is true and the earlier one is false );
```
**if / else / elseif (example)**

**Abort**: causes the job to stop with an execution error and displays information

```gams
parameter P1; P1 = 3;
if (P1 < 0, abort "stopped because P1 < 0", P1);
parameter P2; P2 = 3;
display P2;
```
if / else / elseif (example)

```gams
set a /a1*a3/ ;
parameter P3; P3(a) = ord(a);
*   ---- reassign values
loop(a,
    if ( sameas(a,"a1"), P3(a) = 50;
        else if ( sameas(a,"a2"), P3(a) = 75;
            else if ( sameas(a,"a3"), P3(a) = 100;
                       )
    )
  );
)
display P3;
```
$if / $ifi statements

Execution of a GAMS statement when a conditional is true

Syntax

```gams
$if conditional statement_to_execute
$ifi conditional statement_to_execute
or
$if conditional
    statement_to_execute
```

- $if is case sensitive ($ifi is a case insensitive variant)
- The conditional is evaluated at compile time, so does not involve GAMS calculated numbers
$if / $ifi statements (example)

```plaintext
set a /a1*a3/ ;
parameter p1;

$ifi not defined a display "a is not defined";
$ifi not declared b display "b is not defined";
$ifi not defined p1 display "p1 is not defined";
```

conditionals_if_ifthen.gms
$\text{if} / \$\text{ifi} \text{ statements (example)}$

**conditionals_if_ifthen.gms**

```plaintext
set a /a1*a3/ ;
parameter p1;

$\text{ifi not defined a display "a is not defined"};
$\text{ifi not declared b display "b is not defined"};
$\text{ifi not defined p1 display "p1 is not defined"};
```
$if / $ifi statements (example)

**Abort**: causes the job to stop with an execution error and displays information

* Condition to abort the model in case the base data is missing

```
$ifi not exist "base_data.gdx" $ABORT "base_data.gdx" is missing, in %system.fn%, line %system.incline%
```

**conditionals_if_elseif.gms**
$ifthen / $iftheni statements

Execution of a GAMS statement when a conditional is true

**Syntax**

```
$iftheni
  conditional statement_to_execute
$endif
```

- $ifthen is case sensitive ($iftheni is a case insensitive variant)
- The conditional is evaluated at compile time, so does not involve GAMS calculated numbers
Model attributes and options
Model attributes and options

- Options controlling the content of the LST file
  - $ options
  - Option statements

- Options controlling the solver
  - Model options
  - Model attributes

- [https://www.gams.com/latest/docs/userguides/userguide/u_g__dollar_control_options.html](https://www.gams.com/latest/docs/userguides/userguide/u_g__dollar_control_options.html)
## GAMS language (model attributes)

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelstat</td>
<td>Model status</td>
<td>solvestat</td>
<td>Solver status</td>
</tr>
<tr>
<td>1</td>
<td>Optimal</td>
<td>1</td>
<td>Normal completion</td>
</tr>
<tr>
<td>2</td>
<td>Locally optimal</td>
<td>2</td>
<td>Iteration interrupt</td>
</tr>
<tr>
<td>3</td>
<td>Unbounded</td>
<td>3</td>
<td>Resource interrupt</td>
</tr>
<tr>
<td>4</td>
<td>Infeasible</td>
<td>4</td>
<td>Terminated by solver</td>
</tr>
<tr>
<td>5</td>
<td>Locally infeasible</td>
<td>5</td>
<td>Evaluation error limit</td>
</tr>
<tr>
<td>6</td>
<td>Intermediate infeasible</td>
<td>6</td>
<td>Unknown</td>
</tr>
<tr>
<td>7</td>
<td>Intermediate non-optimal</td>
<td>7</td>
<td>(unused)</td>
</tr>
<tr>
<td>8</td>
<td>Integer solution</td>
<td>8</td>
<td>Error preprocessor error</td>
</tr>
<tr>
<td>9</td>
<td>Intermediate non-integer</td>
<td>9</td>
<td>Error setup failure</td>
</tr>
<tr>
<td>10</td>
<td>Integer infeasible</td>
<td>10</td>
<td>Error solver failure</td>
</tr>
<tr>
<td>11</td>
<td>(unused)</td>
<td>11</td>
<td>Error internal solver error</td>
</tr>
<tr>
<td>12</td>
<td>Error unknown</td>
<td>12</td>
<td>Error post-processor error</td>
</tr>
<tr>
<td>13</td>
<td>Error no solution</td>
<td>13</td>
<td>Error system failure</td>
</tr>
</tbody>
</table>
# GAMS language (model attributes)

## Attributes that can be controlled by the user

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
<th>Default</th>
<th>Global option</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterlim</td>
<td>Iteration limit</td>
<td>1000</td>
<td>iterlim</td>
</tr>
<tr>
<td>limcol</td>
<td>Number of columns displayed for each block of variables</td>
<td>3</td>
<td>limcol</td>
</tr>
<tr>
<td>limrow</td>
<td>Number of rows displayed for each block of equations</td>
<td>3</td>
<td>Limrow</td>
</tr>
<tr>
<td>reslim</td>
<td>Time limit for solver (CPU seconds)</td>
<td>1000</td>
<td>Reslim</td>
</tr>
<tr>
<td>optfile</td>
<td>Option file usage</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>solprint</td>
<td>Solution print option</td>
<td>1</td>
<td>solprint</td>
</tr>
</tbody>
</table>
Model options (example)

```gams
$offlisting
option limrow=18, limcol =0, solprint=off;

option nlp = conopt;

mod.solprint = 0;
mod.iterlim = 0;
mod.optfile = 0;
mod.limrow =0;
```
Setting environment variables

Environment variables

- GAMS recognizes the environment variable GDXCONVERT and GDXCOMPRESS which control the format with which GDX files are written.
  
  - 0  do not compress gdx files (default)
  - 1  compress gdx files

**Syntax**

```
$setenv GDXCOMPRESS number
```
Model development: partial solve
Save and restart

• Feature that allows for running the model in pieces (intermediate work is saved at the end of each run)

• Useful for:
  
  ➢ Separation of model and data
  
  ➢ Model development: by splitting the model in pieces, we can run only the modified ones.
  
  ➢ Running multiple scenarios: This feature can save time when running scenarios and managing results.
Save and restart

Syntax

Gams piece1.gms s=savel
gams piece2.gms r=savel s=save2
gams piece3.gms r=save2

Example

Gams dataHandling.gms s=s1
gams runModel.gms r=s1 s=s2
gams resultReporting.gms r=s2