

GAMS Cheat Sheet

Declarations

GAMS objects have to be declared before their first use. Main objects are

Set	Collection of elements used for indexing. $S = \{a, b, c\}$ is written in GAMS as <code>Set S / a, b, c /;</code> . A sequence of elements, such as <code>t=1990:2010</code> , can be entered as <code>Set t 'Year' / 1900*2010 /;</code> , where 'Year' is an optional explanatory text.
Parameter, Scalar, Table	Exogenous data (given input) to be entered or calculated by the modeler. Scalar is 0-dimensional, Parameter is n -dimensional and Table is a n -dimensional parameter that expects the input in table format.
Variable	Endogenous variables to be determined by GAMS. It is possible to enter the following prefixes before Variable to specify the variable type: Positive , Negative , Binary (variable is 0 or 1), Integer .
Equation	Keyword to define an algebraic relationship between variables.
Model	Collection of equations. To declare a model that includes all the equations: <code>Model model_name / all /;</code> To include a list of equations: <code>Model model_name / eq_name1, eq_name2 /;</code>

Data entry

The general expression to declare and define data is

`data_type symbol_name [symbol_description] [/ symbol_value /];`

Examples:

```
Scalar rho "discount rate" / .15 /;
Parameters b(i) / seattle 20, san-diego 45 /
            salaries(employee,manager,department)
            / anderson .murphy .toy      = 6000
              hendry .smith .toy        = 9000
              hoffman .morgan .cosmetics = 8000 /;
```

Variable attributes

To each variable is associated a series of attributes:

- .l Level of the variable. Receives new values when a model is solved.
- .lo Lower bound (default to -inf).
- .up Upper bound (default to inf).
- .fx To fix a variable (sets the value for the attributes .l, .lo and .up): `x.fx(i) = 1;`
- .m Marginal (or dual) value. Receives new values when a model is solved.

Arithmetic and functions

Arithmetic operations:

`+`, `-`, `*`, `/`, `**` (exponentiation, `x**y` is defined only for $x \geq 0$, if x can be negative, use `power(x, y)` instead).

Most common functions (see the [documentation](#) for the list of intrinsic functions):

`abs()`, `cos()`, `exp()`, `log()`, `log10()`, `max(...)`, `min(...)`, `power(,)`, `round()`, `sin()`.

Relationship operators:

`lt`, `<`, `le`, `<=`, `eq`, `=`, `ne`, `<>`, `ge`, `>=`, `gt`, `>`.

Logical operators:

`not`, `and`, `or`, `xor`.

Special symbols:

`inf` Plus infinity.

`-inf` Minus infinity.

`na` Not available, used for missing data.

`undef` Undefined, result of an undefined operation such as `3 / 0`.

`eps` Numerically equal to zero, but considered as existing. For example, `sum(i $ z(i), 1)` equals 0 if `z(i) = 0` and `sum(i $ z(i), 1)` equals `card(i)` if `z(i) = eps`.

Conditional expressions with dollar condition

Logical expression can be expressed with a dollar condition. For example:

`a $ (b > 1.5) = 2;` means if `b` is greater than 1.5 then `a` equals 2. If `b` is less than-or-equal to 1.5 then the value of `a` remains unchanged.

It can also be used on the right hand side. For example:

`a = 2 $ (b > 1.5);` means that `a` equals 2 if `b` is greater than 1.5, else `a` equals 0.

Indexing

Basic indexing

<code>x(i) = 12;</code>	Assign all elements of <code>x</code> to 12.
<code>b('seattle') = 20;</code>	Assign the element <code>seattle</code> of <code>b</code> to 20.
<code>sum(i, x(i))</code>	Sum <code>x</code> over the set <code>i</code> : $\sum_i x_i$.
<code>sum((i,j), x(i,j))</code>	Sum <code>x</code> over the sets <code>i</code> and <code>j</code> : $\sum_{i,j} x_{i,j}$.
<code>prod(j, y(i,j))</code>	Multiply <code>y</code> over the set <code>j</code> : $\prod_j y_{i,j}$.
<code>Alias(i, j)</code>	Declare that the set <code>j</code> can be used instead of <code>i</code> .
<code>y = smax(i, x(i));</code> or <code>y = smin(i, x(i));</code>	Find the largest or smallest value of a symbol indexed over a set.

Advanced indexing

On ordered sets (for example one defined by `Set t 'Year' / 1900*2010 /;`):

<code>ord(t)</code>	Returns the position of a member in a set: <code>Parameter val(t);</code> <code>val(t) = ord(t);</code> Here <code>val('1900')</code> will be 1, <code>val('1909')</code> 10, and <code>val('2010')</code> 111.
<code>card(t)</code>	Returns the number of elements in a set: <code>card(t)</code> will return 111.
lags and leads	It is possible to use lag or lead operators on ordered sets. For example an equation defining the evolution of capital stock would be: <code>eq.k(t+1) .. k(t+1) =e= (1 - delta) * k(t) + i(t);</code>

`sameAs(r, s)` can be used to test if the active elements of `r` and `s` are the same. For example: `a(r,s) $ (not sameAs(r, s)) = 10;` would assign 10 to all non-diagonal elements of `a`.

It is possible to define subsets: sets whose members must all be members of some larger sets. For example:

```
Sets
  i "all sectors" / light-ind, food+agr, heavy-ind, services /
  t(i) "traded sectors" / light-ind, food+agr, heavy-ind /;
```

Note that a subset is ordered when the indices are entered in the same order as the ordered parent set.

The assignment can also be made dynamically (the set is then called a dynamic set):

```
Set j(i);          Declare j as a subset of i.
j(i) = yes;        Assign all elements of i to j.
j('light-ind') = no; Remove the element 'light-ind' from j.
Or alternatively: j(i) $ (not sameAs(i, 'light-ind')) = yes;.
```

Dynamic subsets present the following restrictions: it is not possible to declare variables defined on dynamic subsets; and they are not ordered, even if their parent sets are.

Equation definition

An equation named `eqname` is defined by

`eqname(index) .. expression eq_type expression ;`

Example: `cost .. z =e= sum((i,j), c(i,j) * x(i,j));`

Main equation types (`eq_type`):

- `=e=` Equality: rhs must equal lhs.
- `=g=` Greater than: lhs must be greater than or equal to rhs.
- `=l=` Less than: lhs must be less than or equal to rhs.

Solve statement

solve *model_name* using *model_type* (maximizing|minimizing *objective_name*);
Example: solve transport using lp minimizing z;
Main model types [*model_type*] ((see the [documentation](#) for the complete list):
cns Constrained Nonlinear System: square system of nonlinear equations, $f(x) = 0$.
lp Linear programming: optimization problem with linear objective and constraints.
mcp Mixed complementarity problem.
mip Mixed integer programming: linear opt. pb. with a mix of continuous and integer variables.
nlp Nonlinear programming: optimization problem with nonlinear objective and constraints.
qcp Quadratic constraint programming: optimization problem with quadratic objective and constraints.

Display

display x, y.1; to ask GAMS to write in the listing file (file with the .lst extension) the value of x and y. For variables, one has to precise the attribute (.1 here).
option decimals = N to restrict the display to the first N decimals.

Flow control

GAMS contains 3 types of loops:
for to loop over a parameter:
 Scalar i;
 for (i = 1 to 1000 by 10,
 display i;
);
loop to loop over a set:
 loop (t, pop(t+1) = pop(t) + growth(t));
while to loop over a general condition:
 Scalar x / 0 /;
 while (x <= 10,
 x = x + 1;
);

Use of the if-else statement:
if (x <= 0,
 y = 1;
elseif (x > 0 and x < 1),
 y = 2;
else
 y = 3;
);
To stop GAMS if a condition is met use abort:
abort \$ (abs(residuals) > 1E-6) "Residual not null", residuals;

Dollar control

Dollar control options can alter GAMS behavior in several ways. The \$ symbol can be placed in the first column or elsewhere on a line if using \$\$ as first two characters. They are executed at compile time, so before any calculation takes place. Most important dollar control options (see the [documentation](#) for the complete list):
\$exit GAMS stop reading the file after \$exit.
\$include Use \$include filename to insert the contents of the file.
\$onText/\$offText Use to enclose severals lines of comments.
\$set Use \$set varname varvalue to define an environmental variable (also called control variable), which value can be accessed later by using %varname%. Additionally, you can set a control variable via a user defined command line parameter option, e.g.,
gams trnsport.gms --varname=varvalue.

Options

Some options can be set using the following syntax:
option *option_name* = *option_value*;
Main options (see the [documentation](#) for the complete list):

<i>option_name</i>	Default	Interpretation
decimals	3	Number of decimals printed.
iterLim	2e9	Limit on the number of iterations used to solve a model.
limCol	3	Control the number of columns (variables) listed at each solve.
limRow	3	Control the number of rows (equations) listed at each solve.
resLim	1000	Limit on the units of processor time used to solve a model.
solver (cns, nlp, lp, ...)	Installation default	Control the solver used to solve a particular model type.

Example: option limCol = 0;

Comments

A line starting with an asterisk '*' is commented:
* This line is a comment
To comment several lines, it is possible to place them between a pair of \$onText/\$offText:
\$onText
Any lines between \$onText and \$offText are commented
\$offText
End-of-line comments can be enabled using \$eolCom followed by the chosen special character:
\$eolCom #
x = 1; # This is an end-of-line comment
In-line comments can be enabled using \$inlineCom followed by a pair of one or two character sequence (default to /* */):
\$inlineCom { }
x { This is an in-line comment } = 1;

GDX files

A GDX file is a binary file that can contains information on sets, parameters, variables, and equations. GDX files are very useful to enter data, to explore results, and to import/export data from various file formats (e.g., csv, Excel, ...).
Compile phase (before any calculation)
\$gdxIn file_name.gdx Open the GDX file for reading.
\$load id1 id2=gdxid2 Read symbols id1 and gdxid2 from the GDX file and assign them to id1 and id2 that have been previously declared.
\$gdxIn Close the GDX file currently open.
Same thing with \$gdxOut and \$unLoad to write data to a GDX file during the compile phase.

Execution phase (after calculations)

execute_load 'file_name.gdx' id1, id2=gdxid2	Read symbols id1 and gdxid2 from the GDX file and assign them to id1 and id2 that have been previously declared.
execute_unload 'file_name.gdx' id1, id2=gdxid2	Write to the GDX file the symbols id1 and id2 and assign id2 to the symbol gdxid2.