Algunos comandos de GAMS

Javier León Caballero
javileon@ucm.es

Universidad Complutense de Madrid

25 de febrero de 2020

Accediendo a elementos

- Los elementos de un conjunto (set) de GAMS son cadenas de caracteres
- Para acceder a un elemento en específico hay que utilizar comillas

```
sets
i ciudades /1*20/;
* ...
eq1..X('1') + X('2') =L= 40;
```

- El tamaño de un conjunto se obtiene usando `card(i)`
- La posición del elemento *i* dentro de su conjunto se obtiene con `ord(i)`

Operador condicional

- A veces querremos hacer sumatorios solo en algunos términos, o definir ecuaciones para algunos índices:

$$\sum_{i|d_i \geq 0} X_i = 100$$

$$\sum_{i} \sum_{j>i} X_{ij} \leq 13$$

$$X_i \geq M Y_i \qquad \forall i > 1$$

Operador condicional

- A veces querremos hacer sumatorios solo en algunos términos, o definir ecuaciones para algunos índices:

$$\sum_{i|d_i \geq 0} X_i = 100$$

$$\sum_{i} \sum_{j>i} X_{ij} \leq 13$$

$$X_i \geq M Y_i \qquad \forall i > 1$$

- **Ojo:** aquello sobre lo que se condicione ha de ser un dato, y no una variable. Si X_i es una variable, lo siguiente no se podrá implementar directamente:

$$\sum_{i|X_i \geq 0} Y_i = 20$$

Operador condicional: $

$$\sum_{i|d_i \geq 0} X_i = 100$$

$$\sum_i \sum_{j>i} X_{ij} \leq 13$$

$$X_i \geq M Y_i \qquad \forall i > 1$$

```
equations
demanda      satisfacer la demanda
eq2
eq3(i)
* El condicional va en la definición, no en la declaración!
;

demanda.. sum(i$(d(i)>=0), X(i)) =E= 100;
eq2..     sum((i,j)$(ord(j)>ord(i)), X(i,j)) =L= 13;
eq3(i)$(ord(i)>1)..  X(i) =G= M*Y(i);
```

Condiciones de terminación

Una ejecución de GAMS (sin fallos) puede terminar por lo siguiente:

- Solución óptima encontrada (*optimal solution*)
- No existe solución factible (*model infeasible*)
- No existe solución acotada (*unbounded*)

Condiciones de terminación

Una ejecución de GAMS (sin fallos) puede terminar por lo siguiente:

- Solución óptima encontrada (*optimal solution*)
- No existe solución factible (*model infeasible*)
- No existe solución acotada (*unbounded*)
- Se han agotado los recursos: memoria, iteraciones, tiempo (por defecto: 1000s)
- En problemas con variables enteras, se ha alcanzado el gap de integralidad (*integrality gap* deseado (por defecto: 0.1)

Condiciones de terminación

¿Qué es el gap?

$P : \min\{c^t x + d^t y \mid Ax + By = b, x \in \mathbb{R}^n, y \in \mathbb{Z}^m, x \geq 0, y \geq 0\}$

- Buscamos el óptimo de P, un problema lineal con variables enteras

Condiciones de terminación

¿Qué es el gap?

$P : \min\{c^t x + d^t y \mid Ax + By = b, x \in \mathbb{R}^n, y \in \mathbb{Z}^m, x \geq 0, y \geq 0\}$

- Buscamos el óptimo de P, un problema lineal con variables enteras
- El solver va a utilizar algoritmos como *branch and bound*, que irá buscando soluciones de dos maneras:

Condiciones de terminación

¿Qué es el gap?

$P : \min\{c^t x + d^t y \mid Ax + By = b, x \in \mathbb{R}^n, y \in \mathbb{Z}^m, x \geq 0, y \geq 0\}$

- Buscamos el óptimo de P, un problema lineal con variables enteras
- El solver va a utilizar algoritmos como *branch and bound*, que irá buscando soluciones de dos maneras:
  1. Soluciones que cumplan las condiciones de integralidad de las y, factibles. Según avance el algoritmo, el valor objetivo de estas soluciones *enteras* irá decreciendo (por estar minimizando)

Condiciones de terminación

¿Qué es el gap?

$P : \min\{c^t x + d^t y \mid Ax + By = b, x \in \mathbb{R}^n, y \in \mathbb{Z}^m, x \geq 0, y \geq 0\}$

- Buscamos el óptimo de P, un problema lineal con variables enteras
- El solver va a utilizar algoritmos como *branch and bound*, que irá buscando soluciones de dos maneras:
  1. Soluciones que cumplan las condiciones de integralidad de las y, factibles. Según avance el algoritmo, el valor objetivo de estas soluciones *enteras* irá decreciendo (por estar minimizando)
  2. Cotas inferiores. Son soluciones a problemas *relajados*. Estas soluciones **no** son necesariamente enteras y por tanto no son factibles de P. Cada paso se van añadiendo más restricciones del problema original, y por tanto los valores objetivos irán aumentando (por estar minimizando)

Condiciones de terminación

¿Qué es el gap?

$P : \min\{c^t x + d^t y \mid Ax + By = b, x \in \mathbb{R}^n, y \in \mathbb{Z}^m, x \geq 0, y \geq 0\}$

- Buscamos el óptimo de P, un problema lineal con variables enteras
- El solver va a utilizar algoritmos como *branch and bound*, que irá buscando soluciones de dos maneras:
  1. Soluciones que cumplan las condiciones de integralidad de las y, factibles. Según avance el algoritmo, el valor objetivo de estas soluciones *enteras* irá decreciendo (por estar minimizando)
  2. Cotas inferiores. Son soluciones a problemas *relajados*. Estas soluciones **no** son necesariamente enteras y por tanto no son factibles de P. Cada paso se van añadiendo más restricciones del problema original, y por tanto los valores objetivos irán aumentando (por estar minimizando)
- GAMS llama a la solución entera *best integer*, y a la cota *best estimate*. Una vez que estos valores coinciden se sabe que se ha llegado al óptimo. Si no:

$$\text{gap} = \frac{\text{best integer} - \text{best estimate}}{\text{best integer}}$$

Condiciones de terminación

Gap: fichero `.log`

```
69            Nodes                                         Cuts/
70       Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap
71
72  *     0+    0                           46.5000                          ---
73  Found incumbent of value 46.500000 after 0.11 sec. (21.83 ticks)
74        0     0       19.6337     8       46.5000       19.6337      251   57.78%
75  *     0+    0                           20.1219       19.6337             2.43%
76  Found incumbent of value 20.121858 after 0.11 sec. (23.97 ticks)
77        0     0       19.6407     9       20.1219       Cuts: 3      263    2.39%
78        0     0       19.6435    10       20.1219    MIRcuts: 2      280    2.38%
79  *     0+    0                           20.0206       19.6435             1.88%
80  Found incumbent of value 20.020616 after 0.13 sec. (39.28 ticks)
81        0     0       19.6446    12       20.0206       Cuts: 2      284    1.88%
82  *     0+    0                            19.7124       19.6446             0.34%
83  Found incumbent of value 19.712394 after 0.13 sec. (45.47 ticks)
```

Figura 1: Ejemplo de gap mostrado por CPLEX/GAMS

Condiciones de terminación

Gap: optcr

```
* Opciones antes de resolver
* Gap relativo tolerado = 0 -> No se parará hasta el óptimo
option optcr = 0;

* 1 hora de tiempo máximo
option reslim = 3600;

model mochila /valorfunc, fobj, dualescenarios/;

solve mochila using mip minimizing OBJ;
```

Sentencia `model`

- Una vez que se han definido las restricciones, construimos un modelo usando la sentencia `model`, incluyendo las ecuaciones que queramos:

```
model mochila /valorfunc, fobj, dualescenarios/;
```

- Es posible usar la palabra `all` para incluir todas las ecuaciones:

```
model mochila /all/;
```

Sentencia `model`

- Un mismo `.gms` puede tener varios modelos definidos, los cuales se pueden unir, o *restar*:

```
model modelo1 /valorfunc, fobj, dualescenarios/;

* Las restricciones de modelo1, junto a una restricción más
model modelo2 /modelo1, otrarestriccion/;

* Las restricciones de modelo2, quitando una restricción
model modelo3 /modelo2-valorfunc/;
```

- Más info:
  `https://www.gams.com/latest/docs/UG_ModelSolve.html`

Los resultados: solve y display

- Una vez definido el modelo, se llama a `solve` para resolverlo:

```
model mochila /valorfunc, fobj, dualescenarios/;

* using MIP: mixed-integer programming
* using LP: linear programming
solve mochila using MIP minimizing OBJ;
```

- Los resultados se ven en el `.lst`. Es cómodo usar `display` para ver los valores de las variables que nos interesen:

```
display OBJ.l, X.l, Y.l;
```

Sobre los sufijos

Para ver el valor de la variable `X` hay que poner `X.l`. Esto es así ya que el valor es solo uno de los atributos de la variable.

```
* Tras el solve: valor (level) de X, y marginal
display X.l, X.m;
```

Hay otros atributos que pueden fijarse antes del `solve`: `.up` (*upper bound*), `.lo` (*lower bound*), `.fx` (*fixed value*)

```
* ANTES del solve
* Para todo i:
X.up(i) = 5;
X.lo(i) = 2;

* Para algún i en específico:
X.up('5') = 7;
X.fx('3') = 3;
```

El atributo `.l` puede ser usado antes del `solve` para fijar un valor inicial

Sobre los sufijos

Para ver el valor de la variable `X` hay que poner `X.l`. Esto es así ya que el valor es solo uno de los atributos de la variable.

```
* Tras el solve: valor (level) de X, y marginal
display X.l, X.m;
```

Hay otros atributos que pueden fijarse antes del `solve`: `.up` (*upper bound*), `.lo` (*lower bound*), `.fx` (*fixed value*)

```
* ANTES del solve
* Para todo i:
X.up(i) = 5;
X.lo(i) = 2;

* Para algún i en específico:
X.up('5') = 7;
X.fx('3') = 3;
```

El atributo `.l` puede ser usado antes del `solve` para fijar un valor inicial

Tipos de variables

Tras definir las variables, se puede establecer un dominio:

- `free` (por defecto): Variable real, sin cotas. Variable objetivo ha de ser de este tipo
- `positive`: real mayor o igual a 0
- `negative`: real menor o igual a 0
- `binary`: 0 o 1
- `integer`: entre 0 y una cota (por defecto, 100)
- Otros tipos: `sos1`, `sos2`, `semicont`, `semiint`. Más info: `https://www.gams.com/latest/docs/UG_Variables.html#UG_Variables_VariableTypes`

```
integer variable X(i);

* Aumentar cota superior
X.up(i) = 500;
```

Licencia de GAMS

GAMS proporciona una licencia de prueba de manera gratuita (previo registro). Sus limitaciones:

- 2000 vars y 2000 restricciones (para problemas LP y MIP)

Conjuntos ordenados: problema

Problema de inventario

Una empresa fabrica un determinado producto, y tiene que satisfacer la demanda durante 4 periodos. En cada periodo ha de decidir cuántas unidades fabricar (cada una a coste cp_t) y cuántas almacenar (a coste ca_t), de manera que se satisfaga la demanda dem_t. No puede almacenar más de *alm_max* unidades cada periodo, y empieza con un inventario de *inv_ini* unidades. Formular el modelo lineal que minimiza el coste.

Conjuntos ordenados: problema

Problema de inventario

Una empresa fabrica un determinado producto, y tiene que satisfacer la demanda durante 4 periodos. En cada periodo ha de decidir cuántas unidades fabricar (cada una a coste cp_t) y cuántas almacenar (a coste ca_t), de manera que se satisfaga la demanda dem_t. No puede almacenar más de *alm_max* unidades cada periodo, y empieza con un inventario de *inv_ini* unidades. Formular el modelo lineal que minimiza el coste.

$$\begin{aligned} \min \ & \sum_t cp_t X_t + ca_t Y_t \\ \text{s.a. } & Y_t = Y_{t-1} + X_t - dem_t \qquad \forall t > 1 \\ & Y_1 = inv\_ini + X_1 - dem_1 \\ & X_t, Y_t \in \mathbb{Z}^{\geq 0} \end{aligned}$$

Conjuntos ordenados: en GAMS

$$\min \sum_t cp_t X_t + ca_t Y_t$$

$$\text{s.a. } Y_t = Y_{t-1} + X_t - dem_t \qquad \forall t > 1$$

$$Y_1 = inv\_ini + X_1 - dem_1$$

$$X_t, Y_t \in \mathbb{Z}^{\geq 0}$$

```
balance(t)$(ord(t)>1).. Y(t) = Y(t-1) + X(t) - dem(t);
balanceini.. Y('1') = inv_ini + X('1') - dem('1');
```

GAMS interpreta `t-1` como el elemento anterior del conjunto ordenado `t`. También se puede usar `t+1` o `t-n`

Conjuntos ordenados: en GAMS

```
balance(t)$(ord(t)>1).. Y(t) = Y(t-1) + X(t) - dem(t);
balanceini.. Y('1') = inv_ini + X('1') - dem('1');
```

Más compacto aún:

```
bal(t).. Y(t) = Y(t-1) + X(t) - dem(t) + inv_ini$(ord(t)=1);
```

- Cuando t vale 1, `Y(t-1)` no está definido, así que GAMS lo interpreta como un 0
- `inv_ini$(ord(t)=1)` suma *inv_ini* solo para $t = 1$, para el resto de t's esto vale 0

Conjuntos ordenados: circulares

- `X(t-1)` se refiere al elemento anterior a `t`. Para el primer elemento el anterior no existe, y vale 0
- Sin embargo hay casos en que los conjuntos pueden ser *circulares*, como por ejemplo estaciones
- En ese caso, el elemento anterior al primero sería el último. Para esto ponemos `X(t--1)`