Introducción al lenguaje GAMS

Jose Ignacio Marín Alberdi

Año 2000

Índice General

1 Acerca de este documento

Este documento se organiza por secciones, cada una de las cuales trata un aspecto concreto relativo al sistema que deseo describir.

La descripción se centra en un comentario acerca del uso general de la herramienta y la gramática básica que define el lenguaje GAMS. Los ejemplos no son demasiado completos pero hay que tener en cuenta que se ha tratado sobre todo de dar las ideas básicas que permitan construir un programa más o menos complejo en GAMS. La sugerencia puede ser examinar los programas de la colección de modelos de GAMS o bien intentar la construcción de algunos ejemplos completos.

Al mismo tiempo este documento puede servir como guia de referencia rápida a la hora de emplear el sistema GAMS, pues resume muchos de los puntos que desarrolla el manual del usuario a lo largo de más de 200 páginas. Sin embargo, aunque se recogen los temas más importantes, este documento no puede sustituir por completo a dicho manual que será necesario a la hora de hacer consideraciones más avanzadas.

Este documento se ha preparado como introducción a la parte de prácticas de optimización con GAMS que actualmente se está dando dentro del Departamento.

Más adelante se puede pensar en otro documento que describa usos más complejos del lenguaje así como casos prácticos interesantes relativos a la industria química implementados por personas de este departamento o tomados de la literatura. O se puede considerar el empleo de los ejemplos desarrollados en CACHE.

Cada vez que mencione un elemento del lenguaje en medio del discurso lo insertaré entre paréntesis. Los ejemplos de código GAMS aparecerán en párrafos centrados independientes. El tipo de letra cambia para enfatizar que se está hablando de código GAMS.

2 ¿Qué es GAMS?

En pocas palabras diré que GAMS es un lenguaje, soportado por un paquete informático, que permite especificar un problema de programación matemática independientemente del método de resolución asociado al mismo. Los problemas a tratar han de ser de naturaleza algebraica no pudiendo especificarse problemas de optimización con ecuaciones diferenciales de forma directa.

Además de poder describir el problema de nuestro interés GAMS puede llamar a los programas encargados de resolver el caso, siempre que estén dentro de su lista de resolvedores disponibles.

La idea detrás de GAMS es la de encapsular la fase de descripción del problema, aislándola de la fase de resolución del mismo. A medida que se desarrollan nuevos programas para resolver determinados tipos de problemas estos pueden ser incorporados al entorno GAMS.

3 ¿Cómo podemos emplear Gams?

Para poder plantear y resolver un problema empleando el sistema GAMS es necesario completar una serie de etapas.

La primera, y no siempre obvia, pasa por tener claro que programa queremos resolver. Es decir tenemos que determinar cuales son las ecuaciones, variables y datos numéricos que manejaremos así como la estructura general del programa. Cuando digo estructura me refiero a los tipos de variables y relaciones entre las mismas que vamos a emplear. Como principales estructuras de programación tenemos disponibles en GAMS las siguientes:

- Programación Lineal (`LP`), todas las variables son de tipo continuo y solamente se permiten relaciones lineales entre las mismas.
- Programación Lineal Entera Mixta (`MILP`), las variables pueden ser de tipo continuo o discreto y solamente se permiten relaciones lineales entre las mismas.
- Programación No Lineal (`NLP`), todas las variables son de tipo continuo y se permiten relaciones lineales y no lineales entre las mismas.
- Programación No Lineal Entera Mixta (`MINLP`), las variables pueden ser de tipo continuo o discreto y se permiten relaciones lineales y no lineales entre las mismas.

Con la primera de estas opciones (`LP`) podemos plantear problemas simplificados o sencillos y alcanzar la solución óptima de una forma bastante eficiente y segura al ser la programación lineal un campo prácticamente cerrado. Las capacidades de modelar con este tipo de programación son menores que en los otros casos, aún así a veces es conveniente contar con versiones (`LP`) orientativas de nuestros modelos.

La siguiente opción en cuanto a riqueza expresiva es la programación lineal entera mixta (`MILP`). Con ella se pueden tratar casos muy complejos de planificación, que tengan en cuenta una casuística compleja. Este tipo de programas es más o menos difícil de resolver dependiendo del número y la influencia de las variables discretas consideradas. Para casos de gran número de variables existen numerosas técnicas heurísticas que permiten encontrar soluciones muy buenas, válidas pese a no ser la óptima en cada caso. Muchas veces, por ser un tipo de programación más sencillo de tratar, los casos que cuentan con partes no lineales son transformados en instancias (`MILP`) para asegurar su resolución, esto es más verdad cuanto mayor sea el caso.

El tipo de programación más evidente a emplear en muchos casos de modelado de sistemas complejos o que estudien fenómenos naturales es la programación no lineal (`NLP`). Sin embargo este tipo de programación es muy compleja y los resolvedores, en el mejor de los casos, pueden encontrar soluciones locales, o incluso soluciones arbitrarias, o a veces no encuentran soluciones cuando estas existen. Para casos de elevado número de variables este tipo de programación es un reto todavía en la actualidad.

La mayor riqueza expresiva, y por ende la mayor dificultad en su tratamiento, la aporta la programación no lineal entera mixta (`MINLP`), que aúna las dificultades del tratamiento no lineal y del tratamiento de posibilidades discretas. Casos de incluso muy reducido tamaño pueden ser casi imposibles de resolver.

Una vez que hemos decidido acerca de como vamos a plantear nuestro caso concreto hemos de ser capaces de escribirlo en lenguaje GAMS. El lenguaje GAMS incorpora

muchos elementos que hacen sencillo el poder dar una descripción de numerosos problemas incluso de grandes dimensiones y elevada complejidad.

Muchas veces, dependiendo de como escribamos el programa los resolvedores tendrán un mejor o peor comportamiento. Hemos de tener esto en cuenta y tender a escribir nuestros programas de forma que el comportamiento del resolvedor sea óptimo en la medida de lo posible.

La última fase, una vez realizadas las pruebas que determinan cual es la mejor forma de escribir nuestro modelo GAMS, consiste en realizar las pasadas oportunas para obtener los resultados deseados.

Una pasada consta de dos partes, una primera en la que GAMS analiza el archivo de datos que hemos preparado con el editor (compilación) y otra, si todo ha ido bien el resolvedor que GAMS tenga que invocar para solucionar el pase lo completa (ejecución).

Normalmente no es fácil interpretar toda la información que se puede obtener de los pases, habitualmente se obtiene mucha información sobre el sistema en sí tratando de ver porque la solución es la que es.

Dependiendo de lo obtenido en la fase de interpretación podemos querer volver a formular el modelo cambiando sus datos o hipótesis de partida hasta conseguir que el resultado se parezca al comportamiento real objeto de estudio.

Como conclusión decir que podemos emplear GAMS principalmente como una herramienta de desarrollo rápido de prototipos de programación matemática.

En otros casos GAMS es una herramienta que participa de lleno en la fase operativa del modelo desarrollado, no solamente en la elaboración del prototipo en sí, pudiendo interaccionar con otros programas de forma sencilla.

4 ¿De que partes consta un programa GAMS?

Las partes más habituales que podremos encontrar en un programa GAMS son las siguientes:

- Cabecera del programa. Aquí se colocan ciertos comentarios descriptivos del programa, título, autores del mismo, información relativa a la versión etc.
- Opciones de programación. Los programadores activan o desactivan ciertas opciones generales de GAMS a su conveniencia. Estas opciones pueden especificar la forma en la que se enseñan los resultados, iteraciones máximas para los algoritmos, declaración de símbolos especiales de comentarios, etc.
- Declaración de conjuntos. En muchas ocasiones los programas requieren que las variables, los parámetros y las ecuaciones sean descritos mediante subíndices. Así, por ejemplo, una ecuación de balance material puede ser requerida para cada elemento del conjunto de platos de una columna de destilación. También podemos tener la variable temperatura definida para cada plato. Es por ello por lo que en esta sección se especifican los diversos conjuntos a tener en cuenta y los elementos que a ellos pertenecen. Ciertos conjuntos pueden ser declarados como combinación de elementos de otros conjuntos. Así podemos tener, por ejemplo, si existen los conjuntos centros y procesos, un nuevo conjunto que diga que procesos consideramos en cada centro, no siendo válidas todas las posibles combinaciones de elementos entre ambos.

- Declaración de parámetros numéricos. Un programa además de tener variables y relaciones entre las mismas, es decir ecuaciones, ha de alimentarse de ciertos valores numéricos constantes que denominaremos parámetros. En su forma más sencillas estos parámetros serán constantes que no se ven afectadas por ningún subíndice. De forma más compleja podemos considerar uno o varios subíndices que afectan a un parámetro determinado. Los datos definidos pueden combinarse mediante expresiones matemáticas para dar lugar a nuevos valores para otros parámetros.

- Declaración de variables. Aquí definiremos que variables vamos a emplear y de que tipo es cada variable. Podremos también fijar los límites de las variables que queramos y especificar valores iniciales a considerar en los pases del resolvedor.

- Declaración de ecuaciones. En esta parte se pueden definir las ecuaciones del modelo. Siempre es necesario definir un tipo especial de ecuación que es la función objetivo, que necesita de la participación de una variable especial que es la que hemos de maximizar o minimizar en el pase de optimización.

- Declaración de modelo. Normalmente a cada programa GAMS le va a corresponder un modelo. Sin embargo, podemos considerar varios modelos dentro de un programa GAMS. Cada modelo está definido por las ecuaciones que se consideran dentro del mismo.

- Lanzamiento del programa de optimización. Esta parte puede ser muy sencilla si solamente queremos lanzar el programa que resuelve el caso una vez. En otras ocasiones podemos querer realizar varias llamadas a diferentes resolvedores con diferentes modelos, realizando operaciones intermedias con los datos o los valores iniciales de las variables, incluyendo bucles y condiciones. Esto puede valer para implementar complejos algoritmos de optimización.

- Informes de resultados. En este caso podemos optar por tener los informes que GAMS prepara de forma automática o bien podemos darle instrucciones para preparar nuestros propios informes en el archivo de salida o en diversos archivos a nuestra conveniencia.

Las diversas partes de un programa GAMS pueden ordenarse como he sugerido o pueden cambiarse de orden a voluntad del programador. Lo único que hay que tener en cuenta es que no se puede usar un símbolo dentro del programa que no haya sido declarado con anterioridad.

Además de todo lo dicho es preciso saber que podemos declarar los conjuntos y parámetros independientemente del momento en el que les asignamos valores, amen de poder incorporar archivos externos en el cuerpo del programa en forma de "includes". Esto permite desarrollar programas que albergan la estructura de un tipo de problema y poder cambiar los datos del mismo de forma externa, tanto los relativos a conjuntos (topología del problema) como las constantes numéricas.

Iremos desarrollando cada una de las partes vistas hasta ahora al mismo tiempo que introducimos la sintaxis del lenguaje que emplea GAMS.

5 Escribiendo en GAMS

Antes de pasar a describir cada una de las partes típicas de un programa GAMS quiero hacer una serie de comentarios acerca de la forma en la que hemos de intro-

ducir los datos en GAMS, que como todos los lenguajes de un pelaje similar tiene sus propias "reglas de ortografia".

Conviene saber que GAMS:

- Acepta solamente archivos de tipo texto plano ASCII, con cualquier extensión, aunque la más empleada sea la extensión "gms".
- No distingue entre minúsculas y mayúsculas.
- Tiene un conjunto de caracteres válidos. Una buena recomendación aquí es olvidarse de tildes, la letra ñ y casi de todo lo que no entre de serie en un teclado americano.
- Tiene una lista de palabras y símbolos reservados del sistema que configuran el lenguaje e iremos viendo poco a poco.
- Las sentencias en GAMS pueden introducirse en formato libre, esto es con blancos y saltos de líneas arbitrarios entre medias. La longitud máxima de una línea es de 120 caracteres.

Sentencias

Suelen empezar con una palabra reservada como (`equation`) o (`parameter`) por ejemplo y normalmente terminan con punto y coma (`;`). Algunas sentencias no comienzan con una palabra clave o reservada, tal es el caso de las sentencias de asignación de valores a constantes numéricas por ejemplo.

Las directivas de compilación, palabras reservadas precedidas por un dolar (`$`) suelen ir en líneas propias que no necesitan terminar en punto y coma (`;`). Veremos algunas a lo largo del texto.

Identificadores

Son los nombres que les ponemos a los conjuntos, parámetros, variables, etc. Para que un identificador sea válido es necesario que comience con una letra y vaya seguido de letras o dígitos hasta completar un máximo de 10 caracteres.

Etiquetas

Son los nombres que damos a los elementos de los conjuntos, pueden tener un máximo de 10 caracteres.

Han de comenzar por letra o dígito y los caracteres empleados a continuación también han de ser letras o dígitos.

Existe otra forma de especificar etiquetas que se basa en el empleo de comillas como delimitadores de las mismas, pero es más conveniente emplear la forma que he mencionado antes.

Textos descriptivos

Los identificadores y las etiquetas pueden ir acompañados de textos descriptivos que el compilador retiene como comentarios a emplear en los archivos de salida.

Este texto puede ir entre comillas o sin comillas. Si se usan las comillas es mejor ya que podemos emplear cualquier carácter en el comentario excepto las comillas (simples o dobles) que empleemos para delimitar el texto. El texto tiene que ir en una línea y no puede superar los 80 caracteres.

Si no se usan comillas, no podemos emplear palabras reservadas al principio del texto, ni separadores como coma (,), punto y coma (;) o slashes (/) en el cuerpo del mismo.

Números

Los números se almacenan en GAMS como valores reales.

GAMS tiene símbolos especiales para infinito (INF), valores no definidos (UNDF), valores muy pequeños (EPS) y valores no disponibles (NA). La palabra (UNDF) aparece siempre como resultado de una operación ilegal como dividir por cero y no puede ser introducida por el usuario como valor numérico.

Normalmente se pueden introducir hasta 10 cifras significativas. Conviene no introducir números mayores que 1.0e+20.

Delimitadores

Los puntos y coma (;) separan sentencias de programa.

La coma (,) separa elementos dentro de un conjunto de datos.

El slash (/) marca el comienzo o el final de una lista de datos.

Comentarios

Los comentarios pueden ser:

Líneas que comienzan con asterisco (*)

Un bloque de texto. Por ejemplo:

```
$ontext
Este es un bloque de texto, aquí puede aparecer cualquier carácter que
queramos,
Por ejemplo la ñ !!!
$offtext
```

El bloque de texto va delimitado por las directivas de compilación adecuadas ($ontext) y ($offtext). Las directivas de compilación aparecen siempre precedidas por el símbolo dólar ($) al principio de línea.

Podemos declarar nuestros propios caracteres como delimitadores de comentarios mediante las siguientes directivas:

```
$eolcom #
$inlinecom {}
x = 1; #ahora este es un comentario
y = 2; {este tambien lo es} z = 3;
```

6 Cabecera del programa

Una vez que sabemos como escribir en GAMS pasaremos a describir las partes más habituales de un programa GAMS.

Normalmente comenzaremos con una directiva ($title) que nos permite introducir el título del pase que estemos preparando. Este título pasará a encabezar las páginas del archivo de salida.

```
$title         PROGRAMA DE EJEMPLO
```

Luego puede venir una sección comentada que de información acerca de la versión, los autores, etc.

```
$ontext
Este programa se ha realizado para que los alumnos de la especialidad
de química tengan una idea de cómo emplear la herramienta GAMS
$offtext
*Autor : Jimba
*
*Version : 1.0
```

Después pondremos una serie de opciones mediante directivas de compilación (`$`) o sentencias de opciones (`OPTIONS`) que nos van a permitir sobre todo especificar como queremos que sea el archivo de resultados que tengamos al final de pasar el programa. También podremos definir nuestros propios símbolos de comentarios u opciones que pasaremos al optimizador de turno.

Las palabras reservadas para dar opciones son (`option`) y (`options`), que como ya sabemos pueden venir en minúsculas o mayúsculas. Las sentencias que empiecen con esta palabra estarán dedicadas a dar valores a ciertas opciones que el compilador pone a nuestro servicio y que hay que conocer. Dentro de una sentencia de opciones podemos dar valores a varios campos siempre que los separemos por comas, espacios o lineas en blanco. Este tipo de sentencias pueden aparecer en cualquier parte del programa, afectando al mismo desde el momento en que aparecen. Como todas las sentencias han de acabar en punto y coma (`;`).

Las opciones que considero de mayor uso son las siguientes, lo que hace cada opción se explica en el comentario del final del trozo de código que viene a continuación:

```
option LIMCOL=0,LIMROW=0;
$offlisting
$onempty
$inlinecom /* */
option solprint=off;
/* las dos primeras evitan que gams desarrolle las ecuaciones y variables
en forma de listado en el archivo de salida
Si quiero ver esos desarrollos para comprobar solo tengo que cambiar
los ceros por otros numeros mas convenientes
la directiva de compilacion offlisting desactiva otra opcion que por
defecto GAMS utiliza y que es la de desarrollar un listado
de los simbolos empleados en el programa en el archivo de salida
la directiva onempty hace que GAMS admita conjuntos vacios
la directiva inlinecom habilita este tipo de comentario
la opcion solprint=off desactiva el listado de las soluciones estandar
de GAMS en el archivo de salida
creo que es mejor decirle que soluciones quiero ver o en algunos casos
escribir las soluciones directamente a otros archivos que no sean el de defecto
```

```
esto ultimo puede venir bien si quiero pasar datos a otros programas
como por ejemplo I LOVE MATLAB para hacer graficas con esos datos
Veremos otras opciones en el transcurso del texto
*/
```

7 Declaración de conjuntos

Los conjuntos se introducen en GAMS dentro de sentencias que empiezan con la palabra clave (`set`) o (`sets`) y acaban en punto y coma (`;`).

El conjunto se define por los elementos que lo forman. Así al nombre del conjunto sigue la lista de sus elementos entre slashes (/). Los elementos del conjunto se separan entre sí por saltos de línea o comas. A los elementos del conjunto, o al conjunto en sí, pueden acompañarles textos descriptivos de los que les separan al menos un carácter en blanco.

```
Sets
i       plantas de proceso      / Madrid, Gijon /
j       ciudades  /  Cordoba
                     Malaga
                     Barcelona /
T       periodos de tiempo /1*5/
;
```

Nótese que en el caso del último conjunto especificado las etiquetas empleadas para los elementos son numéricas. El asterisco (*) viene a especificar un rango en este caso, es decir que consideramos los periodos 1 a 5. Si queremos, para el mejor seguimiento de la información a lo largo del programa aclarar que el subíndice empleado tiene algo que ver con el conjunto "`T`" podemos eliminar la línea de definición de este conjunto y especificar en la misma o en otra sentencia de declaración de conjuntos:

```
SET
T       periodos de tiempo  /t1*t5/
;
```

Creo que es mas claro de esta forma.

Recordad que si queréis especificar algo con unidades, el típico ejemplo de los precios, es útil emplear comentarios entre comillas. Así podemos introducir el slash (/) dentro del comentario sin confundir al compilador.

```
Set


P       "precios de material en euros/tonelada"
/
precseco        precio en la estacion seca
prechumi        precio en la estacion humeda
/;
```

En este ultimo ejemplo, los elementos del conjunto "P" tienen asociado un comentario que no va entre comillas, ojo con poner las tildes en las palabras "estacion" o "humeda", ya que el compilador se os quejará en este caso.

Un subconjunto se declara de igual forma que un conjunto. La única particularidad es que hay que indicar el nombre del conjunto que incluye al que declaramos entre paréntesis a continuación del nombre de este.

```
set jc(j)    ciudades costeras    /Malaga,Barcelona/;
```

Veamos como declarar conjuntos multidimensionales. El ejemplo más sencillo es el bidimensional. En la declaración del conjunto aparecen entre paréntesis los conjuntos de los cuales se toman elementos para establecer las posibles combinaciones. Un elemento del conjunto vendrá dado por un par "(elemento del conjunto 1 . elemento del conjunto 2)", en este caso el punto (.) actua de separador.

```
set    i    /salamanca,cuenca,albacete/;
set    j    /madrid,barcelona,valencia/;
set    ij(i,j)    /salamanca.madrid,albacete.barcelona
                    cuenca.valencia/;
```

Podemos pensar en ejemplos más complicados a medida que incrementamos el número de dimensiones e incluimos la posibilidad de utilizar secuencias. En la siguiente tabla se recoge en la primera columna lo que aparece entre los slashes (/) que sigen a la declaración de un conjunto multidimensional. En la columna de la derecha viene la expansión a elementos sencillos de dicha expresión.

/(a,b).c.d/	a.c.d, b.c.d
/(a,b).(c,d).e/	a.c.e, b.c.e, a.d.e, b.d.e
/(a.1*3).c/	a.1.c, a.2.c, a.3.c
/1*3.1*3.1*3/	1.1.1, 1.1.2, 1.1.3, ..., 3.3.3

En ciertas ocasiones queremos declarar parámetros, variables o ecuaciones que se refieren más de una vez a los elementos de un mismo conjunto. La gramática de GAMS no permite especificar ningún elemento que se refiera directamente varias veces a un mismo conjunto. La única forma de poder hacerlo es mediante el truco de emplear "copias" del conjunto, para que GAMS los tenga por dos conjuntos diferentes pero idénticamente iguales. Estas copias se declaran mediante la sentencia (alias). En el siguiente ejemplo vemos que quiere decir todo esto, y también nos adelantamos un poco a la siguiente sección que habla de la declaración de parámetros.

```
set c    ciudades    /pamplona,sansebas,almeria/;
/*creamos el conjunto c prima que tambien se refiere a las ciudades*/
alias(c,cprima);
*alternativamente podiamos haber dicho alias(cprima,c)
*empleamos el conjunto cprima
parameter
dis(c,cprima)    distancia entre ciudades
;
*dis(c,c) no hubiera sido valido
```

8 Declaración de parámetros

Hay tres tipos de parámetros en GAMS. Por un lado los escalares, asimilables a constantes numéricas, por otro los parámetros, asimilables a listas de constantes numéricas y por último las tablas que son listas de al menos dos dimensiones de constantes numéricas.

Para declarar un escalar o varios hemos de emplear una sentencia que comience con la palabra reservada (`scalar`) o, indistintamente, (`scalars`). Los nombres de los escalares han de comenzar por letra y pueden ir seguidos de letras y números hasta completar 10 caracteres. En la declaración de varios escalares es necesario separar unos de otros mediante saltos de líneas o comas. A continuación de cada nombre de escalar puede venir un texto descriptivo de no más de 80 columnas que necesariamente ha de estar en la misma línea que el símbolo declarado. También puede aparecer, a continuación del texto descriptivo, y delimitado por slashes (/) el valor que toma la constante. Los escalares pueden participar en sentencias de asignación en cualquier momento a partir de su declaración. Notar que una sentencia de asignación no ha de ir precedida de palabra clave alguna.

```
scalars
grav    aceleracion de la gravedad    /9.8/
c        velocidad de la luz        /300000/
r        constante de los gases
ggrav    cuadrado de grav
;
*aqui pueden venir otras sentencias
r=0.082;
ggrav = grav * grav;
```

Para declarar una o varias listas de constantes numéricas asociadas a un índice o conjunto, las reglas son muy similares. La palabra clave a emplear es (`parameter`) o (`parameters`). Al nombre del parámetro le puede seguir un texto descriptivo con las mismas características ya mencionadas en el caso escalar. Los nombres de los parámetros guardan las mismas reglas que todos los identificadores mencionados hasta ahora. Si se dan valores a los elementos de la lista estos han de ir encerrados entre slashes (/). A los elementos de una lista los ha de separar un salto de línea o una coma, cada elemento se define con la etiqueta del elemento del conjunto correspondiente separado del valor que el parámetro adquiera para este índice por un blanco o un signo igual (=). Podemos asignar valores en cualquier momento a partir de la declaración del parámetro. Si asignamos el valor solamente a un elemento concreto de la lista, hemos de indicarlo empleando la etiqueta de ese elemento entre comillas simples en la sentencia de asignación.

```
parameters
presplato (i)     presion en el plato i
volatirel(j)     volatilidad relativa componente j
/
j1 = 5.8
j2 = 1.89 , j3 = 1
```

```
/
;
*la siguiente sentencia afecta a todos los platos
presplato (i) = 14 ;
*y esta afecta a un solo elemento
presplato ('i4') = 14.5 ;
```

Como es natural previamente necesitamos de la declaración de los conjuntos "`i`", platos y "`j`", componentes.

Una asignación interesante que se puede hacer con un parámetro es la de asignar valores distribuidos regularmente sobre un intervalo. Para ello solamente hay que hacer como se dice a continuación. Lo único que hay que tener en cuenta es que (ord) significa ordinal de un conjunto, es decir la posición relativa que un elemento ocupa en el mismo, y (card) su cardinal, es decir el número de elementos totales del mismo.

```
presplato (i) = 14 + (15-14)*(ord(i)-1)/(card(i)-1);
```

Los parámetros pueden llegar a tener hasta 10 dimensiones. Veamos nuevamente un par de ejemplos inventados:

```
set     empleado     /marcos,cecilia,martin/;
set     manager     /luis,rober/;
set     dept     /juguetes,cazaypes,quimica/;
parameter salario(empleado,manager,dept)
/    marcos.luis.cazaypes     90
     cecilia.luis.juguetes    92
     martin.rober.quimica     78
/;
SET     row      /row1*row10/
        col     /col1*col10/ ;
parameter a(row,col)
     / (row1,row4).col2*col7     12
       row10.col10               17
       row1*row7.col10           33
     / ;
```

Para introducir valores en el caso de parámetros multidimensionales las tablas, (`table`), pueden ser una alternativa a las sentencias (`parameter`) basadas en listas, pero en el fondo son más o menos lo mismo. Vemos un par de ejemplos solamente:

```
set     i     "plantas"
     /inchon,ulsan,yosu/
        m     "secciones"
     /atmos-dis,steam-cr,aromatics,hydrodeal/ ;
```

```
TABLE ka(m,i)  "capacidad inicial produccion"
               inchon     ulsan
atmos-dis      3702       12910
steam-cr                  517
aromatics                 181
hydrodeal                 180
+              yosu
atmos-dis      9875
steam-cr       1287
hydrodeal      148
;
```

Las entradas para las que no se da un valor se consideran iguales a 0. Nótese el signo más (+) para indicar la continuación de la tabla. Cuando se tiene más de dos dimensiones es necesario poner, ya sea en las filas o en las columnas, varias de ellas unidas por el punto (.) que ya intervino en la definición de las n-tuplas en la sección de conjuntos. Por ejemplo:

```
table   upgrade (strateg,size,tech)
   small.tech1   small.tech2   medium.tech1   medium.tech2
s1  0.05           0.05          0.05           0.07
s2  0.27           0.24          0.21           0.29
;
```

Una cosa importante es que los datos han de estar alineados correctamente con las etiquetas.

A recordar, si no se dice nada un parámetro o un escalar son inicializados con el valor 0.

9 Declaración de variables

A la hora de declarar variables hemos de considerar de que tipo se tratan. Existen cinco tipos básicos de variables en GAMS: variables libres, variables negativas, variables positivas, variables enteras y variables binarias.

Las palabras clave que aparecen al comienzo de una definición de variables de cada uno de estos tipos son: (`variable`), (`negative variable`), (`positive variable`), (`integer variable`) y (`binary variable`).

Las variables libres son las que están limitadas por menos infinito e infinito, las negativas las que están limitadas por menos infinito y cero, las positivas por cero e infinito. Mientras no se diga lo contrario una variable es continua. Una variable discreta cae dentro del tipo entero, de estas un caso especial son las variables binarias que solamente pueden tener valor de cero o uno.

Para declarar variables de un tipo es necesario emplear la palabra clave que designa ese tipo, a continuación una lista de variables separadas por comas o cambios de

línea y para finalizar el clásico punto y coma (;). La palabra clave puede venir en mayúsculas o minúsculas en singular o plural como viene siendo habitual.

Una variable puede aparecer en varios apartados. Por ejemplo puede aparecer bajo la palabra clave (`variable`) y posteriormente aparecer en una sentencia que empiece como (`positive variable`). Así se va restrigiendo cada vez más el dominio de la variable. Lo normal es introducir cada variable dentro de una sentencia que empiece con las palabras clave de su tipo directamente.

Los nombres de las variables pueden ser combinaciones de hasta 10 letras y números siempre que comiencen por letra. Los nombres de las variables vienen separados por saltos de línea o comas. Los textos descriptivos funcionan igual que en otros casos.

Algunos ejemplos inventados pueden ser:

```
VARIABLE
Beneficio       "cantidad obtenida en euros/mes"
;
positive variables x1,x2,x3;
INTEGER VARIABLES
conecta(i,j)    posibles modulos entre las ciudades i j
;
BINARY VARIABLE
activa1
activa2 ;
```

Observese que "`conecta`" hace referencia a un conjunto de variables de tipo entero. Estas variables, para ser distinguidas unas de otras, vienen afectadas por dos subíndices "`i`" y "`j`". Estos índices podrán tomar valores dentro de lo permitido en las correspondientes definiciones de los conjuntos "`i`" y "`j`".

Cualquier variable dentro de GAMS tiene en todo momento definidos 4 valores asociados o campos. Una vez definida podemos acceder a estos valores ya sea para modificarlos o para consultarlos. Los campos definidos se acceden a través del nombre de la variable seguido de un punto y una palabra clave para el campo. Así, podemos tener:

```
activa1.LO = 0;
activa1.up = 0;
conecta.lo('i1','j5') = 6;
x1.L = 45.5;
costem = x2.m;
```

En los ejemplos vemos que los campos de las variables pueden darse en minúsculas o mayúsculas indistintamente, como es habitual.

En la primera línea fijamos el límite inferior de la variable binaria "`activa1`" al valor 0;

En la segunda línea fijamos el límite superior de la misma variable al valor 0. Estas dos líneas hacen que el único valor posible que puede tomar la variable es 0, con lo cual está fija. Otra forma de especificar esto es empleando el campo auxiliar "`fx`":

```
activa1.fx = 0;
```

En la tercera línea modificamos el límite inferior de la variable que corresponde al subíndice "i1" y "j5" del conjunto de variables enteras llamadas "conecta". Notar que es necesario indicar el campo antes que el elemento en cuestión. Si entre paréntesis encontramos las letras que designan los conjuntos sin los apóstrofes, significará que nos referimos a todas las variables de esa clase en vez de a un elemento concreto.

```
conecta.lo(i,j) = 6;
```

En la siguiente línea, decimos que el valor actual, campo (L) o level, de la variable positiva "x1" toma el valor de 45.5. Si se hace esta asignación antes de invocar a un resolvedor se estará especificando una condición inicial.

Por último, en la siguiente línea, se ve como el parámetro "costem" adquiere el valor del campo (M) o marginal de la variable "x2". Este campo hace referencia al coste marginal de la variable en cuestión.

Las ecuaciones, que bajo cierto punto de vista son totalmente equivalentes a las variables, también tienen definidos estos cuatro campos, pero su utilización no es frecuente en los programas GAMS.

10 Algunas expresiones numéricas

A modo de recetario rápido algunas funciones y expresiones numéricas comunes. Estas expresiones pueden afectar a parámetros o a variables en ecuaciones. En este último caso hay que ver que casi todas introducen no linealidades o discontinuidades que afectarán al tipo de modelo que estemos construyendo. Si estamos haciendo un modelo con ecuaciones lineales (LP) e introducimos una ecuación con la expresión "exp(pres(i))", por poner un ejemplo, pasaremos a tener un modelo no lineal (NLP).

exp(x)	exponencial	power(x,y)	x a la y, con y entero
log(x)	log natural	sqr(x)	cuadrado
log10(x)	log decimal	sqrt(x)	raíz cuadrada
normal(x,y)	num aleatorio según normal media x std. y	sin(x)	seno
uniform(x,y)	num aleatorio según uniforme entre x e y	cos(x)	coseno
abs(x)	valor absoluto	arctan(x)	arc cuya tangente es x
ceil(x)	menor entero mayor o igual que x	round(x)	entero más próximo
floor(x)	mayor entero menor o igual que x	mod(x,y)	resto de x/y
max(x,y,..)	máximo de los argumentos	sign(x)	1 es +, -1 es -, 0 es 0
min(x,y,..)	mínimo de los argumentos	errorf(x)	función error

Sumatorios:

```
sum (i, a(i))
sum ((i,j,k), (a(i,j) * b(j,k)))
```

Productorios: Idem con la palabra reservada (prod) en vez de (sum).

11 Declaración de ecuaciones

Las palabras clave que definen el principio de una sentencia de declaración de nombres de ecuaciones, y que se pueden usar indistintamente, son (`equation`) y (`equations`). En esta sentencia podemos hacer uso de los habituales comentarios descriptivos. Las comas y los saltos de línea separan los nombres de las diversas ecuaciones. Estos nombres han de empezar con una letra que puede ir seguida de más letras y números hasta 10 caracteres.

Un pequeño ejemplo, inspirado en columnas de destilación pero que no es completo, puede bastar para ver las cosas más importantes de esta sección:

```
EQUATION
objetivo     especificacion de los costes de operacion
equil(i,j)   "equilibrio liquido/vapor plato i comp j"
bal(i,j)     balance masa componente j en el plato i
pureza       la pureza requerida en cabeza
*
*faltan bastantes ecuaciones
;
```

Después de dar sus nombres, es necesario decir en que consiste cada ecuación, esto es lo que en GAMS se conoce como definición de ecuaciones. Para especificar cada ecuación tendremos que dar un nombre válido de ecuación, seguir con los caracteres reservados (`..`) y especificar una relación entre variables y parámetros válidos con los operadores normales. En esta relación pueden aparecer funciones y operadores como el sumatorio. La sentencia concluye con el punto y coma (`;`) de rigor.

Las ecuaciones pueden ser de igualdad o de desigualdad. Las igualdades se especifican con el símbolo reservado (`=e=`) o (`=E=`). Las desigualdades mediante el empleo de (`=l=`) o (`=L=`) para las de tipo "menor o igual que" y (`=g=`) o (`=G=`) para las de tipo "mayor o igual que".

Así podemos tener:

```
objetivo .. z =e= Prod - 50 * RR;
equil(i,j) .. y(i,j) =E= x(i,j) * K(i,j);
bal(i,j) .. L(i,j) + V(i,j) =e= L(i+1,j) + V(i-1,j);
pureza .. x('i9','benceno') =g= 0.85;
```

Así en la primera ecuación "`objetivo`" decimos que la variable "`z`" (puede ser el coste de operación) es igual a la cantidad de producto que obtenemos por cabeza (variable "`Prod`") menos 50 veces la relación de reflujo (variable "`RR`").

En las dos líneas siguientes especificamos dos conjuntos de restricciones de igualdad, que se extienden a todas las combinaciones de los elementos de los conjuntos "`i`" y "`j`" (platos y componentes, respectivamente).

El primero de estos conjuntos viene dado por las relaciones de equilibrio líquido vapor. Si tomamos la constante de equilibrio "`K(i,j)`" como un parámetro conocido la relación será lineal, si la declaramos como una variable la relación pasará a ser no lineal. En estas ecuaciones las variables "`x(i,j)`" e "`y(i,j)`" serían las

fracciones molares de componente "j" en el plato "i" en el líquido y en el vapor respectivamente.

En el segundo vemos como se dan los balances por componentes para cada plato. Es interesante ver como podemos hacer que en la ecuación aparezcan variables del plato anterior "V(i-1,j)" o posterior "L(i+1,j)" al considerado en cada caso "i". Las "V" hacen referencia a los flujos de vapor y las "L" a flujos de líquidos.

Como se ve esta ecuación se extendería sobre todas las parejas posibles de elementos de "i" y de "j". Si tenemos en cuenta que se referencian variables de platos posteriores y anteriores al considerado ¿qué pasará cuando GAMS intente escribir la ecuación para el primer o el último plato?. Imaginemos que el conjunto de platos ha sido declarado como:

```
Set
i Platos /i1,i9/ ;
```

Para el primer plato intentará encontrar los elementos "V('i0',j)" dependiendo del componente "j" que estemos considerando, los cuales no existen porque el plato "i0" no existe, y por lo tanto serán reemplazados por un cero.

Para el último plato intentará encontrar los elementos "L('i10',j)" dependiendo del componente "j" que estemos considerando, los cuales tampoco existen porque el plato "i10" no existe, y serán igualmente reemplazados por un cero.

Esto podría arruinar las expresiones de los balances en cabeza y fondo. Por ello es necesario conocer el mecanismo que GAMS tiene para no extender una ecuación o variable a todos los elementos del conjunto que la definen en la sentencia tipo (`equation`). Veremos como se hace esto en la siguiente sección.

La última ecuación no se extiende sobre ningún conjunto ya que afecta solamente a una variable del conjunto de variables "`x(i,j)`" (fracciones molares del componente "`j`" en el liquido del plato "`i`"). Esta variable es "`x('i9','benceno')`", es decir la fracción molar del componente "`benceno`" en el líquido del plato "`i9`". Nótese el mayor o igual en este caso.

En este ejemplo damos por supuesto que se han declarado las variables y parámetros pertinentes, además de los que hicieran falta para completar el modelo. Como creo que es fácil de ver, una declaración tal, que además nos sirve de repaso, hubiera podido ser:

```
SET
i platos /i1*i9/
j componentes /benceno,tolueno,polimero/;
*i1 es el fondo
*i9 es la cabeza
*
* aqui faltaria definir una tabla con las K(i,j)
* si las damos como parametros
*
VARIABLE
z       coste de operacion
```

```
;
POSITIVE VARIABLES
Prod    cantidad de producto obtenido por cabeza
RR      relacion de reflujo
L(i,j)  cantidad de componente j en el liquido del plato i
V(i,j)  cantidad de componente j en el vapor del plato i
x(i,j)  fraccion molar componente j en liquido del plato i
*
*etc... resto de variables necesarias incluyendo y(i,j)
;
*limites de las fracciones molares
x.up(i,j) = 1;
y.up(i,j) = 1;
```

12 Expresiones condicionales con dolar (`$`)

Una primera cosa que hay que saber llegados a este punto es que los símbolos reservados de GAMS para establecer relaciones numéricas entre datos son:

`lt, <`	menor que
`le, <=`	menor o igual que
`eq, =`	igual que
`ne, <>`	no igual que
`ge, >=`	mayor o igual que
`gt, >`	mayor que

Para realizar operaciones lógicas:

`not`	negación
`and`	and
`or`	or inclusivo
`xor`	or exclusivo

Una forma muy empleada en GAMS para trabajar con condiciones lógicas, especialmente para restringir el campo de variación posible de un conjunto, esto es, trabajar con excepciones, es la de emplear la condición dolar (`$`). Normalmente aparece junto a una definición de una ecuación o junto a cualquier término de una asignación, suele ir como "`$(condición)`". Dentro de condición podemos considerar cualquier expresión que no afecte a variables. Los parámetros, escalares, y similares pueden ser empleados aquí, también pueden aparecer campos de variables, esto es "`variable.campo`".

La mejor forma de entender como funciona este operador es mediante una serie de ejemplos:

```
a $(b > 1.5) = 2;
```

"`a`" es igual a 2 siempre que "`b`" sea mayor que 1.5. Si no es así no se hace nada.

```
x = 2 $(y > 1.5) + 5 $(y le 1.5);
```

"x" es igual a 2 siempre que "y" sea mayor que 1.5, si no es así "x" será 5. Si no apareciese el término "+ 5 $(y le 1.5)", entonces "x" sería 0 siempre que "y" no fuese mayor que 1.5.

Veamos el ejemplo que dejamos pendiente de la sección anterior:

```
set i    platos    /i1*i9/;
set ic(i)    cabeza    /i9/;
set ifo(i)    fondo    /i1/;
*notese que no puedo llamar al ultimo subconjunto if(i)
*porque if es una palabra reservada

*ord es ordinal
*card es cardinal
bal(i,j)$(ord(i) gt 1 and ord(i) lt card(i))
        .. L(i,j) + V(i,j) =e= L(i+1,j) + V(i-1,j);

*alternativamente
*bal(i,j)$((not ic(i)) and (not ifo(i)))
*        .. L(i,j) + V(i,j) =e= L(i+1,j) + V(i-1,j);
```

En este caso la expresión de los balances para cada componente solamente aplica a los platos intermedios de la columna, no a los extremos de la misma. Los balances en estos extremos deberían luego definirse como otras ecuaciones aparte. Lo que se ha mostrado en este ejemplo es como limitar el dominio de definición de una ecuación.

Como cosa curiosa hay que notar que cuando el ordinal de un conjunto (ord) es igual a su cardinal (card) estamos hablando del último elemento del mismo.

De la misma forma las condiciones ($) pueden afectar a alguna expresión de las que toman parte en una ecuación, entrando a formar parte de ella efectivamente si se cumple algún requisito necesario.

```
mb(i) .. x(i) =g= y(i) + (e(i) - m(i)) $(t(i));
```

El término "e(i)-m(i)" solamente entra en la ecuación para los elementos del conjunto "i" que también pertenezcan al conjunto "t(i)"

Esto también funciona con conjuntos multidimensionales para mapear una serie de relaciones. En este caso damos valores a un parámetro exclusivamente para ciertas combinaciones válidas de ciudades, el parámetro, claro está, ha tenido que ser declarado previamente, así como los elementos que entran en su definición.

```
set    i    /salamanca,cuenca,albacete/;
set    j    /madrid,barcelona,valencia/;
set    ij(i,j)    /salamanca.madrid,albacete.barcelona
                    cuenca.valencia/;

costenvio(i,j) $(ij(i,j)) = factor * distan(i,j);
```

Por último vemos como extender un operador sumatorio a ciertos elementos exclusivamente de un conjunto.

```
costetot .. coste =e=
        sum ( (i,j) $(ij(i,j)), costenvio(i,j) * x(i,j)) ;
```

Cuando se vean las expresiones de control del flujo del programa se volverán a ver pequeños ejemplos con condiciones ($).

13 Includes

La directiva (`$include`) puede ser muy util a la hora de elaborar programas GAMS. Si el programa GAMS consta de varias partes o módulos cada una de ellas puede ir en un archivo separado y pueden ser reunidas en un archivo principal mediante el empleo de esta directiva.

Lo que hace la directiva es incluir el archivo cuyo nombre se especifica a continuación de la misma justo en el punto en el que aparece dentro del archivo que se está compilando.

Un ejemplo de buen uso de la directiva (`$include`) es el de hacer un programa GAMS que declare toda su estructura de conjuntos, parámetros, variables y ecuaciones, así como la forma de resolver el pase en un archivo principal a compilar.

```
set j ciudades;
parameter
pob(j)    poblacion en ciudad j  ;
*variables etc...
```

Los datos relativos a que elementos aparecen en cada conjunto y datos numéricos de valores de parámetros pueden aparecer en otros archivos que serán incluidos en el principal antes de invocar al resolvedor o tener que mencionar un elemento concreto de un conjunto.

```
*antes de que sea necesario
$include "conj.dat"
$include "data.dat"
*resto del programa
```

Así, solamente hemos de cambiar los archivos auxiliares si queremos cambiar los datos numéricos del problema o la topología (estructura que definan los conjuntos) del mismo.

14 Declaración de modelos

Supongamos que estamos en el caso más sencillo de solamente querer considerar un modelo en nuestro programa GAMS. Para especificar cual es este modelo lo primero que hemos de hacer es pensar un nombre para el mismo.

Imaginemos que llamamos a nuestro modelo "`prueba`". Normalmente si solamente hay un modelo en el programa, suele incluir todas las ecuaciones que hemos especificado en el mismo. Así la sentencia para declarar nuestro modelo "`prueba`", el cual ha de contener a todas las ecuaciones del modelo es:

```
model prueba /all/;
```

Las palabras (`model`) y (`all`) son palabras reservadas del sistema.

Como nombres válidos podemos considerar asociaciones de letras y números hasta 10 caracteres que empiecen con una letra. Podemos acompañar la definición con un texto explicativo que no exceda las 80 columnas y que esté contenido en la misma línea de declaración.

```
MODEL prueba2 Esta es una prueba con comentario /all/;
```

Si queremos declarar varios modelos dentro de un mismo programa hemos de decir que ecuaciones intervienen en cada uno de ellos.

Supongamos ahora que tenemos las ecuaciones "`ecu1`", "`ecu2`", "`ecu3`", "`ecu4`", y que vamos a declarar un modelo "`completo`" que las incluya a todas, otro, modelo "`parte1`", que incluya solamente a las ecuaciones "`ecu1`", "`ecu2`" y "`ecu3`", y por último un modelo "`parte2`" que solamente considere las ecuaciones "`ecu1`", "`ecu3`" y "`ecu4`". Las sentencias GAMS que describen esta situación son:

```
model completo /all/;
model parte1 /ecu1,ecu2,ecu3/;
model parte2 /ecu1,ecu3,ecu4/;
```

Observese que en cada modelo es necesario que aparezca una función objetivo, que no tiene porque ser la misma en cada caso. En nuestro ejemplo "ecu1" puede ser la función objetivo que sería la misma para los tres modelos definidos.

15 Lanzamiento del programa de optimización. Invocando a GAMS

Nos falta ver que instrucción activa el proceso de optimización.

La sentencia que empieza por la palabra reservada (`solve`) cumple exactamente ese cometido. En la sentencia es necesario especificar el modelo que se pretende resolver, además del tipo de programa del que se trata (`LP, NLP, MILP, MINLP`) y que variable queremos maximizar o minimizar, es decir cual es el objetivo.

El orden en el que se den estos dos últimos parámetros no es importante.

Un par de ejemplos:

```
Solve prueba1 using lp minimizing z;
Solve completo maximizing obj2 using minlp;
```

La variable a maximizar o minimizar ha de ser continua y de tipo libre, esto es no puede ser ni positiva ni negativa, y al menos ha de participar en una ecuación de las que componen el modelo.

A partir de esta instrucción GAMS realiza un paso de compilación, verifica los límites de las variables (que los inferiores no excedan a los superiores) y que no se realicen operaciones no permitidas como divisiones por cero. Luego el control pasa al programa optimizador encargado de resolver cada tipo de modelo. Por defecto GAMS sabe a que programa acudir, pero el programador siempre puede disponer del resolvedor de su elección siempre que este pueda atacar el tipo de problemas del que se trate. Para ello, antes de una sentencia (`solve`):

```
Option lp=minos5;
```

Esta sentencia dice que los modelos lineales pasan a ser competencia del resolvedor minos, en vez de la opción por defecto que pudiera existir en aquel momento, por ejemplo osl.

A través de ciertos sufijos el programador puede controlar o acceder a ciertas opciones relativas al modelo. Las más interesantes, los comentarios explican sucintamente lo que cada una significa, pueden ser:

```
Model prueba /all/;
*PONER un limite maximo de iteraciones
prueba.iterlim=5000;
*ESTABLECER archivo de opciones para el resolvedor
*algunas opciones solamente pueden ser dadas de esta forma
prueba.optfile=myoptionfile;
*ESTABLECER criterio absoluto de terminacion
prueba.optca=0;
*ESTABLECER criterio relativo de terminacion
prueba.optcr=0.1;
*ESTABLECER MBytes a emplear por el resolvedor
prueba.workspace=500;
*PARA preguntar por el estado del modelo
*prueba.modelstat, este puede dar lo siguiente
*1 solucion optima, 2 optimo local,
*3 ilimitado, 4 infactible
*5 local infactible, 6 intermedio infactible,
*7 intermedio suboptimo, 8 solucion entera,
*9 intermedio no entero, 10 entero infactible
*PARA preguntar por el numero de ecuaciones generadas
*prueba.numequ
*PARA preguntar por el numero de variables generadas
*prueba.numvar
```

Una vez editado el archivo de texto y guardado este es necesario invocar al programa ejecutable GAMS que ha de hacer el pase de compilación y el pase de ejecución si el paso anterior no ha dado ningún error.

Si se ha instalado GAMS en el sistema lo normal es que su directorio raiz este en definido en el PATH. Si no es así deberemos incluir una línea en el script de inicio de sesión, que bajo el sistema operativo Windows es el autoexec.bat, que se encargue de esta cuestión. Si suponemos que el sistema GAMS está en el directorio c:\gams, en Windows esa línea sería:

```
PATH=c:\gams;%PATH%
```

Al iniciar la máquina el operativo tendría actualizada la variable de entorno PATH que dice donde se han de buscar los archivos requeridos por línea de comandos. A partir de ahora suponemos que estamos trabajando en Windows.

Una vez comprobado que invocando GAMS desde una ventana DOS el sistema responde, pasamos a decirle que procese nuestro archivo. Lo adecuado en este punto es trabajar en un directorio propio de trabajo, que bien pudiera ser "c:\trabajo\prueba". Imaginemos que nuestro archivo se llama "prueba.txt". Así las cosas la forma más facil de invocar a GAMS y decirle lo que tiene que hacer es:

```
>: gams prueba.txt wdir=c:\trabajo\prueba
```

De esta forma le decimos que archivo procesar y en que directorio trabajar. Es la forma más limpia y cómoda. Así todos los archivos que genere GAMS se encontrarán bajo el directorio que le hemos marcado, los archivos que necesite (si hemos especificado algún include) serán buscados en este directorio.

Normalmente la gente emplea la extensión ".gms" para los archivos GAMS, esta es la extensión que el ejecutable de GAMS busca por defecto. Si nuestro archivo se llama "prueba.gms" la instrucción anterior puede resumirse a:

```
>: gams prueba wdir=c:\trabajo\prueba
```

El archivo de salida por defecto de GAMS tiene el mismo nombre que nuestro archivo de entrada y tiene la extensión ".lst". Si hemos lanzado el ejecutable de GAMS con la instrucción anterior, al finalizar el proceso aparecerá el archivo "prueba.lst" bajo el directorio "c:\trabajo\prueba".

En este archivo GAMS deposita:

- Una copia del listado de entrada, para poder verificar ciertos errores. Aparece siempre. En esta listado, allí donde se produzca un error de compilación aparecerá un simbolo dolar (`$`) seguido de un código numérico bajo la línea en cuestión. Al final de la copia del listado de entrada se explica lo que cada código quiere decir. Esto vale como ayuda a la hora de depurar un programa.
- Una lista de símbolos declarados. Conviene desactivarla con (`$offlisting`).
- El desarrollo de las variables y ecuaciones del modelo, limitado por las opciones (`limcol`) y (`limrow`). Para comprobar algunas definiciones basadas en conjuntos conviente no poner estas opciones a 0, cuando vemos que todo va bien, lo mejor es si hacerlo.

- Estadísticas del modelo, número de variables, ecuaciónes, tiempo de generación, cálculo, estados finales del resolvedor y del modelo, etc. Interesa verlo.
- Listado de la solución. Primero el valor del objetivo, luego el listado de las ecuaciones, con sus campos (`.lo,.up,.l,.m`), y luego lo mismo para las variables. Si una ecuación o variable no cumple sus límites se marca con la palabra (`INFES`), si una variable participa en una dirección de crecimiento ilimitado del objetivo se marca con la palabra (`UNBD`). Si queremos eliminar este listado hemos de hacer "`option solprint=off;`".

16 Sentencia Display. Informes de resultados

Hemos visto que si no especificamos nada GAMS genera un archivo de resultados bastante completo en el que nos muestra la solución obtenida en el pase. Mucha de la información vertida en este archivo puede ser desactivada a nuestra voluntad empleando las pertinentes opciones como hemos comentado con anterioridad.

Si hemos podido eliminar todo vestigio de información que GAMS pudiera poner en este archivo también hemos de tener la capacidad de mostrar información acerca de los elementos definidos en nuestro programa a nuestro criterio.

La sentencia más sencilla para lograrlo es la sentencia que comienza con la palabra reservada (`display`). En esta sentencia, a continuación de la palabra reservada, aparecen los nombres de parámetros o campos válidos (`.l,.m,.lo,.up`) de ecuaciones y variables, separados por comas o por saltos de líneas. El resultado es que en el archivo de salida, a la altura correspondiente a la sentencia (`display`) aparece el desarrollo de lo que hayamos solicitado. No es necesario indicar los conjuntos de los que depende el símbolo a desarrollar. Un ejemplo sencillo:

```
*ggrav es un escalar, x es una variable que depende de (i,j)
display ggrav, x.l;
```

Así podemos seleccionar de forma sencilla que información tenemos al finalizar el proceso.

Otro punto interesante, sobre el que no nos extenderemos en absoluto, es la posibilidad de escribir la información que queramos a un archivo que no sea el normal que emplea GAMS como salida. Rápidamente, sobre un ejemplo:

```
*tras invocar el resolvedor
*abrir archivo
file resul /resul.dat/  ;
*escribir en resul
put resul  ;
/*
la siguiente linea escribe la palabra Resultados
un slash es un cambio de linea
*/
put 'Resultados'//;
/*
```

```
bucle para escribir varios elementos
la arroba seguida de unñumero indica a partir de que columna se escribe
el sufijo .tl indica el texto de etiqueta del elemento del conjunto
dado
los dos puntos tras el elemento a mostrar indica el formato numerico
a emplear
8 de ancho total con 4 decimales
*/
loop((i,j),
put  i.tl,  @12,  j.tl  @24,  x.l(i,j):8:4/
);
*cerrar archivo
putclose resul  ;
```

Existen muchas opciones dedicadas a controlar como se muestra la información en los archivos de salida. Si alguien quiere profundizar en estas cuestiones puede consultar el capítulo 14 del manual de GAMS disponible en el Departamento.

17 Elementos de programación

Las estructuras de control del flujo del programa son las sentencias (`loop`), (`if`), (`while`) y (`for`).

Como veremos en algún ejemplo las sentencias que lanzan la optimización (`solve`) pueden aparecer dentro de este tipo de sentencias de control de flujo. Lo que no puede ir dentro de este tipo de sentencias son las declaraciones de elementos del programa, como variables, parámetros, ecuaciones, etc.

También veremos que es posible tomar acciones diferentes según los resultados de la optimización. Para ello hemos de preguntar por el valor del campo (`.modelstat`) del modelo que hayamos definido.

loop ((conjuntos) $(condición) , sentencias);

```
loop (i $(sub(i)),
d(i)=1.5+aumento;
aumento=aumento+0.5;
);
loop ((i,j) $(ord(i) eq ord(j)),
a(i,j)=1;
);
```

En el primer caso ejecutamos las sentencias en la intrucción (`loop`) solamente en caso que el elemento del conjunto "`i`" correspondiente entre en la definición del subconjunto "`sub(i)`" del conjunto "`i`". "`aumento`" es un escalar que tenemos que haber definido previamente.

En el segundo, miramos que los ordinales de los conjuntos "`i`" y "`j`" coincidan. Esta sentencia hace lo mismo que:

```
a(i,j) $(ord(i) eq ord(j)) = 1;
```

if (condicion, sentencias else sentencias);

if (condición, sentencias elseif (condición, sentencias););

```
if ( (mymodel.modelstat eq 4),
*mymodel era infactible
*vamos a probar a relajar los limites de una variable
    x.up(j) = 2*x.up(j) ;
    solve mymodel using nlp minimizing z ;
    else
        if ( (mymodel.modelstatñe 1),
            abort "error al resolver mymodel";
            );
    );
```

En este ejemplo tomamos diversas acciones así haya transcurrido la optimización. En el último (`if`) si no obtenemos una solución al problema lanzamos una sentencia (`abort`) que para la ejecución de GAMS lanzando un mensaje de error.

while (condicion, sentencias);

```
scalar count /1/;
scalar globmin;
globmin = inf;
while ( count le 1000,
    x.l(j)=uniform(0,1);
    solve prueba using nlp minimizing obj ;
    if (obj.l le globmin,
        globmin = obj.l;
        globinit(j) = x.l(j) ;
    );

count = count+1;
);
```

En este ejemplo hemos visto como invocamos al optimizador (`solve`) tantas veces como permite la condición de la sentencia (`while`). Si la solución obtenida en una pasada es mejor que la que teníamos apuntada, actualizamos los parámetros que nos sirven para llevar la cuenta. No hay que olvidar incrementar el contador "`count`" si queremos que el bucle tenga un final.

for (i= start to end by incremento, sentencias);

```
for (s= -3.4 to 0.3 by 1.4,
display s;
);
```

Este ejemplo tan sencillo vale para ver que los valores de inicio y final no tienen porque ser enteros. Tampoco tiene que serlo el incremento. La única restricción es que el incremento sea un número real positivo. En el ejemplo anterior de la sentencia (`while`) puede hacerse casi de idéntica forma con una sentencia (`for`).

18 Un ejemplo completo y la librería de modelos de GAMS

El siguiente es un ejemplo tomado del manual de GAMS, es tan sencillo que a estas alturas debe ser autoexplicativo:

```
$title Modelo sencillo de transporte
option limcol=0,limrow=0;
$offlisting;
option solprint=off;
SETS
   i    plantas de envasado     / Seattle, San-Diego /
   j   mercados           / Washington, Chicago, Topeka / ;

PARAMETERS
        A(i)    capacidad de la planta i
        /       Seattle      350
                San-Diego    600
        /
        B(j)    demanda del mercado j
        /
                Washington    325
                Chicago       300
                Topeka        275
        / ;

TABLE D(i,j)    distancia en miles de millas
                Washington    Chicago    Topeka
    Seattle     2.5           1.7        1.8
    San-Diego   2.5           1.6        1.4
;

Scalar   F coste de envio por mil millas /90/;

PARAMETER C(i,j)     coste transporte en miles de dolares;
C(i,j) = F * D(i,j) / 1000;

VARIABLES
    x(i,j)    cantidades enviadas
    z         precio total transporte ;
POSITIVE VARIABLE x;
```

```
EQUATIONS
    cost         define la funcion objetivo
    supply(i)    respetar capacidad planta i
    demand(j)    respetar demanda mercado j  ;

cost .. z =e= sum((i,j), C(i,j)*x(i,j))  ;
supply(i) .. sum(j, x(i,j)) =l= A(i)  ;
demand(j) .. sum(i, x(i,j)) =l= B(j)  ;

model transport /all/;
SOLVE transport using LP minimizing z;
display x.l,x.m;
```

Para ver este y otros ejemplos más completos con detalles puede ser de utilidad el consultar la libreria de modelos que GAMS incorpora.

La libreria es invocada con el comando "gamslib" seguido del nombre del problema a estudiar o, alternativamente, del número que este tenga asignado dentro de la colección. Por último puede aparecer un nombre de archivo sobre el que queramos que GAMS copie el problema de la colección. Si no decimos nada este archivo sera "prob[número].gms".

Para ver el índice de la colección se puede hacer:

>: gamslib index

Algunos de los ejemplos de la colección son relativos a casos de la industria química.

19 Más allá de lo explicado aquí

Aunque en el texto se han tratado más aspectos que los mencionados en el capítulo de introducción a GAMS del manual aún nos hemos dejado bastantes cosas en el tintero como cabe suponerse.

Así por ejemplo, en este documento no he explicado como preparar algoritmos complejos de resolución con GAMS. Estos algoritmos pueden necesitar del concurso de elementos más avanzados del lenguaje que los explicados hasta aquí ahora.

Un ejemplo de lo que digo puede ser implementar la resolución de un programa MILP mediante una técnica llamada descomposición de Benders, la cual necesita que intervenga un tipo especial de conjuntos en GAMS que se denomina dinámico.

Tampoco he dado listados de casos de interés dentro del sector de la industria química, ni he explicado nada de las técnicas de optimización que corren por debajo del nivel externo de GAMS.

Todo ello puede quedar para documentos posteriores.