Capítulo 10

La herramienta GAMS

10.1 Introducción

GAMS [53] (General Algebraic Modeling System) es un lenguaje de programación que permite el modelado, análisis y resolución de diversos problemas de optimización. Aunque inicialmente el manejo y comprensión de sus estructuras requiere cierto esfuerzo, una vez entendidas se dispone de una herramienta muy versátil capaz de resolver problemas de programación matemática. A pesar de ser una magnífica herramienta, el lector debe ser consciente de las limitaciones impuestas por el estado del arte existente en programación matemática.

Otros lenguajes similares a GAMS son AMPL [41, 54] y AIMMS [13, 55]. Todos ellos presentan características análogas y, en general, no hay razón alguna para elegir uno u otro. En este libro se opta por GAMS dada la familiaridad de los autores con este lenguaje.

Entre las características más importantes de GAMS cabe destacar

1. Su capacidad para pasar de resolver problemas de pequeña dimensión (docenas de variables y restricciones) a problemas mayores (miles de variables y restricciones) sin variar el código sustancialmente. El manejo eficiente de sus índices permite escribir de manera compacta restricciones similares mediante una sola restricción.

2. Separa el proceso de modelado del proceso de resolución del problema. Así, el usuario de GAMS debe ser capaz de conseguir una formulación consistente del problema, y una vez la expresa en la notación de GAMS, este lenguaje hace uso de alguno de los optimizadores disponibles para obtener su solución. De esta manera, el usuario sólo ha de centrarse en obtener un modelo del problema y puede ignorar el funcionamiento interno del algoritmo que se necesita para resolverlo. La separación de estas dos tareas permite cambiar el modelo para mejorarlo o completarlo cómodamente.

3. La forma en que GAMS representa un problema de optimización coincide, prácticamente, con la descripción matemática de ese problema. Por

tanto, el código GAMS es sencillo de comprender para aquellos lectores familiarizados con la optimización.

4. Además, GAMS proporciona los mecanismos necesarios para resolver problemas de optimización con estructuras similares, como son aquellos que se derivan de las técnicas de descomposición.

El usuario de GAMS debe ser cuidadoso con las reglas "gramaticales" de GAMS. El incumplimiento de una sola de ellas puede provocar muchos errores de compilación.

Entre la bibliografía de este lenguaje de programación cabe destacar el manual de GAMS [53], cuyo segundo capítulo ofrece un resumen con las características principales para empezar a programar en este lenguaje, y el artículo [24], que proporciona un enfoque ingenieril de GAMS.

En lo que sigue, se analizará un ejemplo para mostrar las características básicas de este lenguaje.

10.2 Ejemplo ilustrativo

Se ha elegido el problema del transporte para resaltar las características principales de GAMS. Este problema se describe a continuación.

Considérese un conjunto de consumidores y productores. Conocida (1) la demanda de cada consumidor, (2) la producción máxima de los productores, y (3) el coste del transporte de los productos desde los productores hasta los consumidores, el problema del transporte consiste en determinar la cantidad de producto a transportar de forma que el coste sea mínimo.

La formulación del problema anterior de programación matemática es como sigue. Minimizar

$$\sum_i \sum_j c_{ij} x_{ij}$$

sujeto a

$$\begin{aligned} \sum_j x_{ij} &\leq a_i, \ \forall i \\ \sum_i x_{ij} &\geq b_j, \ \forall j \\ x_{ij} &\geq 0, \ \forall i, j \end{aligned}$$

donde los subíndices son

i: para los productores

j: para los consumidores

Los datos son

a_i: la producción máxima del productor i en toneladas

b_j: la demanda del consumidor j en toneladas

Tabla 10.1: Distancias en kilómetros entre productores y consumidores

	Consumidores		
Productores	m_1	m_2	m_3
p1	2.0	1.6	1.8
p2	2.5	1.2	1.4

c_{ij}: el coste de transporte desde el productor i al consumidor j en euros por tonelada y kilómetro

Las variables de decisión son

x_{ij}: la cantidad de producto que se transporta desde el productor i hasta el consumidor j en toneladas

Considérense dos productores y dos consumidores. Las producciones máximas son 300 y 500 toneladas. Las demandas son de 100, 200 y 300 toneladas. Las distancias en kilómetros entre productores y consumidores vienen dadas en la Tabla 10.1.

El coste de transporte en euros es de 0.09 por tonelada y kilómetro.

En GAMS, el problema anterior se codifica y almacena en un fichero cuya extensión por defecto es '.gms'. El contenido de este fichero es el siguiente (el lector no debe tratar de comprender todo el código en esta primera aproximación, ya que a lo largo de este capítulo se explicarán en detalle las reglas del lenguaje; basta con adquirir una visión global de este fichero de entrada GAMS):

```
$Title  El Problema de Transporte  * Un ejemplo de transporte sencillo

  Sets
       i   productores   / p1, p2 /
       j   consumidores   / m1*m3 /;

  Table d(i,j)  distancia en kilómetros
                m1        m2         m3
      p1        2.0       1.6        1.8
      p2        2.5       1.2        1.4;

  Scalar f  coste del transporte (euros por tonelada y km)  /0.09/;

  Parameters
       a(i)  produccion maxima del productor i en toneladas
         /    p1    300
              p2    500  /
       b(j)  demanda del consumidor j en toneladas
         /    m1    100
              m2    200
              m3    300  /
```

```
     c(i,j)  coste de transporte en euros por tonelada;

c(i,j) = f * d(i,j);

Variables
     x(i,j)  cantidad de producto transportada entre i y j en toneladas
     z       coste total del transporte en euros;

Positive Variable x;

Equations
     coste       funcion objetivo
     maximo(i)   cumplimiento de maxima produccion del productor i
     demanda(j)  cumplimiento de la demanda del consumidor j;

coste ..        z  =e=  sum((i,j), c(i,j)*x(i,j));
maximo(i) ..    sum(j, x(i,j))  =l=  a(i);
demanda(j) ..   sum(i, x(i,j))  =g=  b(j);

Model transporte /all/;

Solve transporte using lp minimizing z;

Display x.l;
```

El lector puede comprobar con este ejemplo la similitud entre la formulación matemática y el código GAMS; concretamente, en la parte donde se definen las restricciones (la definición comienza en `coste ..`).

En la primera línea del código anterior se da nombre al problema, y en la segunda se escribe un texto a modo de comentario. En GAMS, cualquier línea que comienza con el símbolo "∗" (asterisco) se interpreta como un comentario.

El comando `Sets` se emplea para definir dos estructuras típicas de GAMS: los índices, usados posteriormente para recorrer los vectores del problema, y los rangos de valores entre los que pueden variar estos índices. En el ejemplo propuesto, el índice `i` se refiere a los posibles productores `p1` y `p2`. De igual manera, el índice `j` se refiere a los posibles consumidores de `m1` a (representado por un asterisco) `m3`.

El comando `Table` permite definir matrices de datos (como puede ser la matriz de la Tabla 10.1 en el ejemplo). Obsérvese la facilidad con que se pueden representar las tablas en GAMS.

El comando `Scalar` es necesario en la declaración y asignación de escalares (como el escalar `f` en el ejemplo).

El comando `Parameters` se utiliza para declaración y asignación de vectores de datos. En este ejemplo se emplean dos vectores: `a(i)` y `b(j)`, que se refieren a la producción máxima, y a la demanda, respectivamente. La asignación mediante este comando se realiza para cada elemento del vector, por lo que es necesario especificar en primer lugar la posición del vector y en segundo, el valor a asignar.

Para declarar la matriz correspondiente al coste de transporte, `c(i,j)`, también se utiliza el comando `parameter`, sin embargo, la asignación de los elementos se realiza posteriormente como función de otros datos.

Las variables del problema de optimización se declaran mediante el comando `Variables`. Sólo cuando se completa la ejecución del código GAMS el optimizador determina el valor de estas variables. En GAMS se debe especificar el carácter de las variables, en este caso, se indica que `x(i,j)` es una variable positiva (no negativa).

El comando `Equations` permite indicar el nombre con que se referenciará a las restricciones del problema, incluida la función objetivo. Tras el comando, aparece el nombre en primer lugar; si es necesario indicar que existen varias restricciones con la misma estructura, se indica seguidamente mediante el índice correspondiente entre paréntesis. Una vez declaradas las restricciones con este comando, se pasa a su definición colocando el símbolo "`..`" y su formulación correspondiente en formato GAMS tras el nombre con el que se han declarado. Muchas funciones matemáticas se pueden emplear en la definición de las restricciones, como suma y multiplicación, entre otras.

El siguiente paso consiste en asociar un nombre al modelo y especificar qué conjunto de restricciones lo forman mediante el comando `Model`. En el ejemplo, el modelo se denomina `transporte`, y está formado por todas las restricciones previamente definidas (lo indica la palabra `all`).

Por último, se emplea el comando `Solve` para que GAMS llame al optimizador pertinente que finalmente resuelve el problema (en el ejemplo, la palabra `lp` indica que se debe usar un optimizador para programación lineal).

De manera opcional, se puede ordenar a GAMS que muestre el valor de los resultados más interesantes en un fichero de salida. Aquí, por ejemplo, el comando `Display x.l` tiene como objetivo mostrar el valor obtenido para el vector de variables `x(i,j)`.

Una vez que se ha escrito el código GAMS en el fichero de entrada, se compila; si no existen errores de sintaxis se resolverá el problema mediante el optimizador correspondiente y se escribirá la solución en un fichero de salida con extensión ".lst". En este fichero, aquellos valores de la solución que son cercanos a cero se representan mediante `EPS`, y aquellos que son cero se denotan mediante un punto.

Puesto que en el código del ejemplo propuesto se ha evitado cometer errores, en el fichero de salida se escribe la solución que ofrece el optimizador. Algunas de las partes que componen este fichero se muestran a continuación.

```
---- EQU MAXIMO      cumplimiento de maxima produccion del productor i
      LOWER     LEVEL     UPPER    MARGINAL
p1     -INF    100.000   300.000       .
p2     -INF    500.000   500.000    -0.036

---- EQU DEMANDA     cumplimiento de la demanda del consumidor j
      LOWER     LEVEL     UPPER    MARGINAL
m1   100.000   100.000     +INF      0.180
```

```
m2   200.000   200.000     +INF      0.144
m3   300.000   300.000     +INF      0.162
```

En esta parte del fichero se muestran los valores mínimo (`LOWER`), óptimo (`LEVEL`), máximo (`UPPER`), y el coste marginal (`MARGINAL`) asociado a las restricciones denominadas `MAXIMO` y `DEMANDA`.

Los valores máximos y mínimos que se asocian a una restricción dependen de su carácter. Así, si la restricción es del tipo "mayor o igual que" (como son las restricciones `SUPPLY`), el valor mínimo es $-\infty$ (`-INF` en el fichero de salida GAMS), y el valor máximo es el término independiente de esa restricción. Por otro lado, si la restricción es de "menor o igual que" (como son las restricciones `DEMANDA`), el valor mínimo es su término independiente, y el máximo es $+\infty$ (`+INF` en el fichero de salida GAMS). Si se tiene una restricción de igualdad, los valores máximo y mínimo coinciden con el valor de su término independiente.

El valor óptimo de una restricción se obtiene al evaluar su término dependiente una vez que se ha resuelto el problema.

Al igual que el valor óptimo, el coste marginal se obtiene como resultado de la optimización. Este valor representa la variación que experimenta el valor de la función objetivo frente a un cambio en el término independiente de la restricción (valor de su variable dual).

En el fichero de salida de GAMS también se presentan los valores que toman las variables de optimización tras la resolución del problema. A continuación, se muestra esta información.

```
---- VAR X cantidad de producto transportada entre i y j en toneladas
         LOWER     LEVEL     UPPER    MARGINAL
p1.m1      .      100.000     +INF       .
p1.m2      .         .        +INF       EPS
p1.m3      .         .        +INF       .
p2.m1      .         .        +INF      0.081
p2.m2      .      200.000     +INF       .
p2.m3      .      300.000     +INF       .

                       LOWER     LEVEL     UPPER    MARGINAL
---- VAR Z              -INF     77.400     +INF       .
  Z           coste total de transporte en euros
```

Se observa como para cada variable se obtiene su valor mínimo, óptimo, máximo y marginal. Si en el fichero de entrada no se especificó el límite inferior (superior) de la variable, entonces se considera que este valor es $-\infty$ ($+\infty$). Los valores óptimo y marginal de una variables se determinan una vez resuelto el problema.

Tabla 10.2: Algunos comandos de la herramienta GAMS

Comando	Objetivo
Set(s)	Declara un(os) conjunto(s) de índices y los propios índices de los vectores
Scalar(s)	Declara los escalares y, opcionalmente, les asigna un valor
Parameter(s)	Declara los vectores de datos y, opcionalmente, les asigna valores
Table(s)	Declara y asigna valores a las matrices de datos
Variable(s)	Declara las variables de optimización y su carácter proporcionándoles una cota superior e inferior
Equation(s)	Declara las restricciones y la función objetivo del problema
Model	Declara los modelos y las restricciones que los componen
Solve	Indica a GAMS que utilice un optimizador determinado para resolver el modelo
Display	Indica qué resultados deben ser presentados en el fichero de salida de GAMS

10.3 Características del lenguaje

En los siguientes apartados se describen los principales comandos de GAMS. En el manual de GAMS [53] se puede completar esta información.

Para resolver un problema usando GAMS es necesario crear un fichero de entrada donde se escribe la formulación del mismo con formato GAMS. Este fichero se divide en varios bloques y cada bloque se dedica a la representación de una parte del problema. En la Tabla 10.2 se muestran los distintos bloques (mediante los comandos asociados) de un fichero de entrada GAMS.

Antes de utilizar GAMS, es conveniente conocer sus reglas. A continuación, se describen las más importantes.

1. En GAMS, es indiferente el uso de mayúsculas o minúsculas.

2. Cada comando debe terminar con un punto y coma. El olvido de este separador de órdenes puede provocar muchos errores de compilación.

3. Los comandos pueden aparecer en cualquier lugar siempre que los datos y variables que se utilizan hayan sido declarados con anterioridad.

4. Como en cualquier otro lenguaje de programación, los identificadores que se utilicen para declarar datos o variables no pueden coincidir con las palabras reservadas de GAMS.

5. En algunos comandos de GAMS, el uso de la "s" final es indiferente. Así, se puede emplear indistintamente el comando `Set` o `Sets` para definir uno o varios conjuntos de índices.

6. Es posible escribir varios comandos en una misma línea siempre que estén separadas por punto y coma. El compilador considera sucesivos espacios en blanco como uno solo.

7. Un simple comando es válido para declarar o definir uno o varios elementos del mismo tipo. No hay necesidad de repetir el nombre del comando.

8. Para documentar el código GAMS es necesario incluir líneas de comentarios. El compilador de GAMS ignora cualquier línea cuya primera columna es un asterisco, considerándola un comentario. También es posible añadir un texto aclaratorio (opcionalmente entre comillas) tras algunos comandos de declaración en GAMS, como son: `set`, `scalar`, `parameter`, `table`, `equation`, y `model`. Este texto debe ayudar a comprender la utilidad del elemento.

9. La mayoría de los comandos son necesarios en la declaración de elementos y, opcionalmente, para asignarle valores. Declarar un elemento (escalar, vector, matriz, etc...) consiste en asignarle un identificador mediante cualquiera de los comandos declarativos (sets, scalar, parameter, table, variables, equations, y model). La declaración permite que un elemento pueda utilizarse en comandos posteriores.

10. Los identificadores usados en GAMS deben comenzar por una letra y pueden ir seguidos por hasta nueve caracteres alfanuméricos, no estando permitida la letra ñ, ni los caracteres especiales como los acentos (esto último tampoco está permitido en los textos explicativos).

10.3.1 Conjuntos

La palabra reservada `Set` o `Sets` se utiliza en GAMS para declarar un conjunto de índices y los propios índices que se emplearán para referirse a los componentes de un vector. Normalmente, la declaración del conjunto va acompañada de la relación de índices que lo componen. En el siguiente ejemplo se declaran los conjuntos `i` y `j`.

```
Sets
     i   productores   / p1, p2 /
     j   consumidores   / m1*m3 /;
```

El compilador de GAMS considera que el texto (`productores` y `consumidores`), que sigue al identificador del conjunto, es un comentario. Los elementos del conjunto, o posibles índices, se especifican entre dos símbolos "`/`" . Cuando los índices forman una secuencia ordenada de letras o números se puede utilizar un asterisco entre el primero y último elemento de la secuencia, evitando escribir cada uno de ellos. Por ejemplo, `/m1*m3/` equivale a `/m1, m2, m3/`. No obstante, GAMS no considera los índices como números sino como caracteres. El final de la declaración de los conjuntos puede terminar con un punto y coma.

Los conjuntos descritos anteriormente son conjuntos constantes o estáticos, es decir sus elementos no varían durante la ejecución del programa. En GAMS, también es posible definir conjuntos dinámicos que deben estar formados por elementos de conjuntos estáticos, previamente definidos. Los conjuntos dinámicos son útiles para definir subconjuntos cuyos elementos cambian en tiempo de ejecución. Dada la complejidad de estos conjuntos, se les dedica el apartado 10.3.12. Por otro lado, se puede establecer una aplicación que asocia elementos de conjuntos ya definidos. Como ejemplo, a continuación se representa una red de nudos definiendo una aplicación de dos elementos con las conexiones entre nudos.

```
Sets N       conjunto de nudos      /N1*N4/
     AP(N,N) conexion entre nudos   /N1.N2,N1.N3,N2.N4,N3.N4/
```

El código anterior define una red de 4 nudos y 4 conexiones. Para comprobar si dos nudos, n y n') están conectados se puede emplear el siguiente código (véase la Tabla 10.8 para más detalles de la sentencia `if`):

```
Alias(N,NP);
If(NP$AP(N,NP), ...; );
```

El comando `alias` es necesario en el caso anterior para referirse a distintos nudos dentro del mismo conjunto. Con este comando se consigue que se pueda usar indistintamente el identificador `N` o `NP` para referirse al conjunto de nudos (no se duplica el conjunto). Típicamente, los alias se usan en operaciones como sumas, productos, etc., donde se referencian dos o más índices del mismo conjunto y se necesita que varíen independientemente.

Para operar con elementos de conjuntos se definen en GAMS los siguientes operadores: `card`, `ord`, `+`, `-`, `++` y `--`, teniendo en cuenta que las operaciones sólo son válidas sobre conjuntos ordenados. Por conjunto ordenado se entiende aquel conjunto estático que está compuesto por una secuencia ordenada de índices. Existen distintas formas para conseguir que un conjunto estático sea ordenado:

- Si sus índices no pertenecen a ningún otro conjunto previamente declarado
- Si sus índices a pesar de pertenecer a otro conjunto previamente declarado y ordenado mantienen la misma posición relativa que en ese conjunto

A continuación se muestra un ejemplo con conjuntos ordenados y desordenados.

```
Sets A1 /Martes, Miercoles, Viernes, Jueves/
     A2 /Domingo, Jueves, Sabado, Viernes/
     A3 /Miercoles, Viernes, Sabado, Lunes/;
```

En este ejemplo, el conjunto `A1` está ordenado ya que sus elementos no pertenecen a ningún otro conjunto previamente definido. Sin embargo, el conjunto `A2` está desordenado porque la posición relativa entre el `Jueves` y el `Viernes` es distinta de su posición relativa en el conjunto `A1` definido con anterioridad. El último conjunto está ordenado porque la posición relativa de sus elementos es

Tabla 10.3: Operaciones sobre el conjunto ordenado `A1`

Operación	Resultado
`card(A1)`	4
`ord('Miercoles')`	2
`'Miercoles'+1`	Viernes
`'Jueves'++1`	Martes
`'Martes'--1`	Jueves

la misma que en el conjunto ordenado `A1`, y es independiente de que la posición relativa del `Viernes` y el `Sabado` en `A2` sea distinta, pues este conjunto está desordenado.

Para comprender mejor el funcionamiento de los operadores, en la Tabla 10.3 se muestra el resultado de las operaciones sobre el conjunto `A1`. El operador `card` devuelve el número de elementos que forman el conjunto (este operador es válido para conjuntos dinámicos o estáticos desordenados). El operador `ord` devuelve la posición que ocupa un elemento dentro del conjunto ordenado. Dado un elemento de un conjunto ordenado, es posible referirse al elemento inmediatamente anterior o posterior por medio de los operadores `+1` o `-1`. Si se usan estos operadores, no se considera que el conjunto es una lista circular de elementos, es decir, no existe el siguiente al último, ni el anterior al primero de sus elementos. Los operadores `++` y `--` permiten el tratamiento de conjuntos ordenados como listas circulares; por tanto, el siguiente del último es el primero, y el anterior al primero es el último elemento.

10.3.2 Escalares

En GAMS los escalares se declaran, y opcionalmente pueden iniciarse, usando el comando `Scalar` o `Scalars`. Se pueden incluir comentarios entre el identificador y el valor inicial, como se ilustra a continuación.

```
Scalar f  coste de transporte (euros por tonelada y km)  /0.09/;
```

Se declara el escalar `f` y se le asigna el valor inicial 0.09 que aparece entre barras "`/`". Con el punto y coma se pone fin a este comando declarativo.

10.3.3 Vectores y matrices de datos

Los comandos `Parameter` y `Table` se utilizan en GAMS para definir vectores de varias dimensiones. Estos comandos se pueden usar indistintamente, salvo para definir vectores de una sola dimensión. En estos casos, sólo se puede usar el comando `Parameter`. En este apartado, se explica, en primer lugar, el comando `Parameter`, en segundo, el comando `Table` y por último, se comparan ambos.

Para declarar vectores de datos se pueden utilizar las palabras reservadas `Parameter` o `Parameters` indistintamente. La asignación de valores iniciales es opcional, al igual que los comentarios aclaratorios.

```
Parameters
     a(i) produccion maxima del productor i en toneladas
       /    p1    300
            p2    500  /;
```

En el caso anterior, el parámetro `a(i)` tiene como índices aquellos que pertenecen al conjunto `i`. Para cada índice de este vector (`p1,p2`), se tiene un valor inicial distinto (`300,500`). La asignación de los valores inciales está comprendida entre dos barras "`/`". El punto y coma pone fin al comando. Para iniciar vectores en GAMS se deben tener presentes las siguientes reglas:

1. Las parejas de índice y valor inicial deben ir entre barras y separadas por comas o en distintas líneas.
2. Estas parejas pueden escribirse en cualquier orden.
3. El valor por defecto que GAMS asigna a los vectores de datos es cero. Por tanto, sólo es necesario especificar los valores distintos de cero.
4. Los índices deben pertenecer al conjunto con que se ha declarado el vector, si no se producirán errores en la compilación.

No siempre es necesario asignar un valor inicial a los vectores con este comando, la asignación se puede hacer posteriormente, como ocurre en el siguiente caso.

```
Parameter c(i,j)  coste del transporte en euros por tonelada;
          c(i,j) = f * d(i,j);
```

Obsérvese la capacidad que tiene GAMS para realizar una asignación escribiendo muy poco código. `c(i,j)` es una matriz que se ha definido como el producto de un escalar `f` ya definido, y otra matriz `d(i,j)`.

La asignación anterior es la versión compacta del siguiente código:

```
Parameter c(i,j)  coste del transporte en euros por tonelada
            /  p1.m1  0.180
               p2.m1  0.225
               p1.m2  0.144
               p2.m2  0.108
               p1.m3  0.162
               p2.m3  0.126  /;
```

Los vectores con más de dos dimensiones se pueden declarar indistintamente con los comandos `Parameter` o `Table`. Las palabras reservadas `Table` o `Tables` permiten definir un vector de dos o más dimensiones. Después del identificador del vector, en la sentencia de declaración pueden incluirse comentarios, como ocurre en el siguiente caso

```
Table d(i,j)  distancia en km
              m1        m2        m3
    p1        2.0       1.6       1.8
    p2        2.5       1.2       1.4;
```

En este ejemplo, la matriz de datos `d(i,j)` tiene como índices `i` y `j`. Los valores iniciales (`2.0, 1.6, 1.8, 2.5, 1.2, 1.4`) de la matriz se asignan a cada una de las posiciones (`p1.m1, p1.m2, p1.m3, p2.m1, p2.m2, p2.m3`).

Para concluir este apartado, se establece la equivalencia entre los dos comandos descritos para definir vectores. El siguiente código define el vector `c(i,j)` con el comando `Table`.

```
Table c(i,j)  coste de transporte en euros por tonelada
           m1     m2     m3
    p1     0.180  0.144  0.162
    p2     0.225  0.108  0.126;
```

Obsérvese que lo anterior no es la manera más compacta para representar la matriz `c(i,j)` en GAMS.

Para definir vectores de más de dos dimensiones considérese el siguiente ejemplo. Se quiere representar en un vector la cantidad de personas que coinciden en su religión, raza y nivel de estudios. El código para representar este vector en GAMS puede ser:

```
Sets re tipos de religion /re1*re3/
     sl niveles de estudios /sl1*sl2/
     ra tipos de razas /ra1*ra4/;

Table c(re,sl,ra)  cantidad de personas con coincidencias
                    ra1        ra2      ra3     ra4
    re1*re2.sl1      2          4       10       2
    re3.sl1          6          2        5       7
    re1*re3.sl2      3          9        7       8 ;

Display c;
```

Este código declara e inicia un vector de tres dimensiones de manera compacta (el funcionamiento del asterisco se explica en el apartado 10.3.9). Con el comando `Display` se escribirá la información siguiente en el fichero de salida GAMS.

```
 ----     11 PARAMETER C   cantidad de personas con coincidencias
                ra1         ra2         ra3         ra4
re1.sl1       2.000       4.000      10.000       2.000
re1.sl2       3.000       9.000       7.000       8.000
re2.sl1       2.000       4.000      10.000       2.000
re2.sl2       3.000       9.000       7.000       8.000
re3.sl1       6.000       2.000       5.000       7.000
re3.sl2       3.000       9.000       7.000       8.000
```

10.3.4 Reglas sobre las expresiones matemáticas en asignaciones

Se deben tener en cuenta las siguientes reglas a la hora de realizar asignaciones mediante el uso de expresiones matemáticas:

1. Es posible con una sola sentencia asignar valores a vectores y matrices, sin necesidad de recorrer mediante bucles cada uno de sus elementos. Por ejemplo, la siguiente asignación asigna un 3 a todos los elementos de la matriz c_{ij}.

```
c(i,j)=3;
```

2. Si se requiere asignar un valor a una posición concreta de la matriz c_{ij} se deben indicar los índices que la caracterizan entre comillas, como ocurre a continuación

```
c('p1','m1')=0.180;
```

3. Los valores asignados mediante los comandos Scalars, Parameters, y Tables pueden modificarse mediante una sentencia de asignación. Los valores que prevalecen son los correspondientes a la última asignación escrita.

4. Una expresión matemática puede contener cualquier operación y/o función matemática estándar. La Tabla 10.4 muestra algunas de las funciones matemáticas disponibles en GAMS. Por ejemplo, para representar un vector aleatorio $x_i, \forall i = 1 \cdots 10$ que sigue una distribución uniforme $U(0, 0.5)$, se puede usar el siguiente código:

```
Set I /I1*I10/; Parameter x(I);
x(I)=uniform(0,1)*0.5;
```

10.3.5 Variables

Las variables en GAMS se declaran de la siguiente manera:

```
Variables
     x(i,j)  cantidad transportada en toneladas
     z       coste de transporte en euros;
```

Las palabras reservadas `Variable` o `Variables` se utilizan indistintamente para declarar las variables a optimizar. Al declarar las variables de optimización es necesario especificar su dimensión. La variable que contendrá el valor de la función objetivo una vez resuelto el problema siempre debe ser declarada en GAMS (como la variable `z` en el ejemplo del transporte). Se pueden incluir comentarios tras el identificador de la variable. La declaración finaliza con un punto y coma.

Cuando se declaran las variables se puede, opcionalmente, indicar su naturaleza:

```
Positive Variable x;
```

Tabla 10.4: Algunas funciones matemáticas en GAMS

Función	Descripción
abs(x)	Valor absoluto de x
arctan(x)	Inversa de la función tangente (en radianes)
ceil(x)	El menor valor entero mayor o igual que x
cos(x)	Función coseno (x en radianes)
errorf(x)	Función de distribución
	de la distribución normal $N(0,1)$ en x
exp(x)	Función exponencial
floor(x)	El mayor valor entero menor o igual que x
log(x)	El logaritmo natural de x
log10(x)	Logaritmo en base 10 de x
mapval(x)	Función aplicación
max(x_1,x_2,...)	Máximo de la lista
min(x_1,x_2,...)	Mínimo de la lista
mod(x,y)	Resto de la división de x entre y
normal(x,y)	Número aleatorio obtenido de una variable
	normal de media x y desviación típica y
power(x,y)	Función potencial x^y (donde y debe ser entero)
$x**y$	Función potencial x^y (donde x debe ser positivo)
round(x)	Redondeo de x al valor entero más cercano
round(x,y)	Redondeo de x con y posiciones decimales
sign(x)	Signo de x, 1 si positivo, -1 si negativo, y 0 si cero.
sin(x)	Función seno (en radianes)
sqr(x)	Cuadrado de x
sqrt(x)	Raíz cuadrada de x
trunc(x)	Equivale a sign(x) * floor(abs(x))
uniform(x,y)	Número aleatorio con distribución uniforme $U(x,y)$

Lo anterior establece una cota inferior para la variable `x(i,j)`, esto es, $x_{ij} \geq 0$. En este caso, no es necesario indicar de nuevo la dimensión de la variable, puesto que cuando se declaró ya se hizo.

Además, es posible declarar variables sin límites (opción por defecto) mediante el comando `Free`, variables binarias mediante `Binary`, enteras con `Integer`, o negativas con `Negative`. La variable que representa el valor de la función objetivo debe declararse sin límites usando el comando `Free`.

En la Tabla 10.5 se presentan los distintos tipos de variables y el intervalo de variación asociado.

El intervalo de variación que GAMS asigna por defecto se puede modificar mediante sufijos sobre las variables. Por ejemplo, la siguiente expresión matemática

$$2.0 \leq r \leq 5.0$$

Tabla 10.5: Tipos de variables e intervalo de variación

Tipo de variable	Intervalo	Intervalo por defecto
Binary	$\{0, 1\}$	$\{0, 1\}$
Free (default)	$(-\infty, \infty)$	$(-\infty, \infty)$
Integer	$\{0, 1, \ldots, n\}$	$\{0, 1, \ldots, 100\}$
Negative	$(-\infty, 0)$	$(-\infty, 0)$
Positive	$(0, \infty)$	$(0, \infty)$

se escribe en GAMS como

```
Positive variable r;
r.lo = 2.0;  r.up = 5.0;
```

donde el sufijo `lo` indica la cota inferior de la variable, mientras que el sufijo `up` denota su cota superior. Con estos sufijos se puede modificar, por ejemplo, el valor límite por defecto de una variable, como ocurre en el siguiente caso que se cambia el límite superior de una variable entera `i` de 100 a 1000:

```
Integer variable i; i.up=1000;
```

En el caso de querer fijar el valor de una variable, se puede asignar el mismo valor a su cota inferior y superior, o utilizar un nuevo sufijo `fx`. Por ejemplo, la expresión

$$y_i = 3.0, \quad \forall i$$

se escribe en GAMS como

```
y.fx(i) = 3.0;
```

Obsérvese que con una única sentencia GAMS se puede asignar el mismo valor a todos los elementos de un vector, independientemente de su dimensión.

Cuando se plantean problemas de programación no lineal es conveniente establecer el punto inicial de algunas variables. Por ejemplo, la siguiente asignación toma como punto inicial para la variable s_{ij} el valor 3.0 (no se congela la variable). Así la siguiente asignación

$$s_{ij} \leftarrow 3.0, \quad \forall i, \forall j$$

puede hacerse en GAMS como sigue

```
s.l(i,j) = 3.0;
```

el sufijo `l` obliga al optimizador a tomar el valor asignado como punto inicial.

10.3.6 Restricciones

Las palabras reservadas `Equation` o `Equations` se utilizan en GAMS para declarar restricciones y la función objetivo de problemas de optimización. Se pueden declarar tanto restricciones de igualdad (ecuaciones) como de desigualdad.

La declaración de restricciones en el ejemplo de transporte se muestra en las líneas 2–4 a continuación. Una vez declaradas, las restricciones se definen utilizando dos puntos entre su identificador y la definición (véanse las líneas 5–7). Para modelar las restricciones se utiliza la notación matemática estándar. La definición debe finalizar con un punto y coma.

```
Equations
     coste       funcion objetivo
     maximo(i)   cumplimiento de maxima produccion del productor i
     demanda(j)  cumplimiento de la demanda del consumidor j;

coste ..        z  =e=  sum((i,j), c(i,j)*x(i,j));
maximo(i) ..    sum(j, x(i,j))  =l=  a(i);
demanda(j) ..   sum(i, x(i,j))  =g=  b(j);
```

Obsérvese que el sumatorio `sum(i, x(i,j))` en GAMS indica $\sum_i x_{ij}$. Al igual que el sumatorio, la operación producto $\Pi_i x_{ij}$ se representa en GAMS como `prod(i,x(i,j))`.

La definición de las restricciones en GAMS debe incluir una de las siguientes secuencias:

- `=e=` para indicar que la restricción es de igualdad,
- `=l=` para indicar que la restricción es de "menor o igual que"
- `=g=` para indicar que la restricción es de "mayor o igual que".

Si al declarar una restricción se incluyen conjuntos de índices tras su identificador significa que esta restricción representa en realidad un bloque de restricciones con idéntica estructura. El número de restricciones que representa el bloque viene dado por la dimensión de los conjuntos con que se declaren. Así, la siguiente definición de la restricción `maximo(i)`

```
maximo(i) ..    sum(j, x(i,j))  =l=  a(i);
```

equivale a las siguientes restricciones

```
maximo1 ..   sum(j,x('p1',j))  =l=  a('p1');
maximo2 ..   sum(j,x('p2',j))  =l=  a('p2');
```

El lector puede comprobar la capacidad de GAMS para compactar información.

10.3.7 Modelos

La palabra reservada `Model` se utiliza para identificar qué conjunto de las restricciones definidas forman parte de un modelo. En el ejemplo presentado, este comando asegura que se incluyan en el modelo denominado `transporte` todas (`all`) las restricciones definidas.

```
Model transporte /all/;
```

Lo anterior equivale al siguiente comando.

```
Model transporte /coste,maximo,demanda/;
```

Para un mismo conjunto de restricciones es posible definir varios modelos que contengan un subconjunto de ellas. Por ejemplo, el siguiente modelo `transporte1` no incluye las restricciones de `demanda`.

```
Model transporte1 "No incluye restriccion de demanda" /coste,maximo/;
```

Obsérvese que se han incluido comentarios entre comillas tras el identificador del modelo.

10.3.8 Resolución

La palabra reservada `Solve` se utiliza en GAMS para resolver el problema definido. En el ejemplo, este comando indica a GAMS que resuelva el modelo `transporte` usando un optimizador de programación lineal (`lp`) que minimize el valor de la variable `z`.

```
Solve transporte using lp minimizing z;
```

Además de la opción `lp`, utilizada en programación lineal, GAMS ofrece otras opciones como se indica en la Tabla 10.6.

Tabla 10.6: Opciones disponibles para la resolución de problemas en GAMS

Opción	Tipo de problema
`lp`	Programación lineal
`nlp`	Programación no lineal
`dnlp`	Programación no lineal con derivadas discontinuas
`mip`	Programación lineal entera-mixta
`rmip`	Programación lineal relajada entera-mixta
`minlp`	Programación no lineal entera-mixta
`rminlp`	Programación no lineal relajada entera-mixta
`mcp`	Problemas complementarios mixtos
`mpec`	Problemas matemáticos con restricciones de equilibrio
`cns`	Sistemas no lineales acotados

Una vez que se resuelve un problema con GAMS es posible extraer información de utilidad mediante algunos sufijos sobre el identificador del modelo resuelto. Entre los sufijos más importantes destacan: `modelstat` que informa sobre la calidad de la solución obtenida, `solvestat` que informa sobre el estado del optimizador al finalizar la resolución, y `resusd` que ofrece el tiempo de cálculo (expresado en segundos) que ha empleado el optimizador en resolver el problema. En el apartado 10.3.10 se presenta un ejemplo que usa el sufijo `resusd`. El siguiente ejemplo muestra el empleo del sufijo `modelstat`.

Tabla 10.7: Posibles valores para los sufijos `modelstat` y `solvestat`

Valor	`modelstat`	`solvestat`
1	Optimal (`lp`, `rmip`)	Normal completion
2	Locally optimal (`nlp`)	Iteration interrupt
3	Unbounded (`lp`, `rmip`, `nlp`)	Resource interrupt
4	Infeasible (`lp`, `rmip`)	Terminated by solver
5	Locally infeasible (`nlp`)	Evaluation error limit
6	Intermediate infeasible (`nlp`)	Error
7	Intermediate non-optimal (`nlp`)	-
8	Integer solution (`mip`)	-
9	Intermediate non-integer (`mip`)	-
10	Integer infeasible (`mip`)	-
11	–	-
12	Error unknown	-
13	Error no solution	-

```
If (transport.modelstat ne 1,
     Solve transport1 using lp minimizing z;
);
```

El código anterior comprueba en primer lugar si la solución obtenida para el modelo `transporte` es óptima mediante la condición `if` (se explica más adelante en este capítulo) y el sufijo `modelstat`. Si la solución no es óptima, se resuelve el modelo `transporte1` con menos restricciones. El sufijo `solvestat` se emplea de igual manera.

Según sea la naturaleza (lineal, no lineal, entera ...) del problema de optimización resuelto, el valor que devuelve GAMS al emplear los sufijos `solvestat` y `solvestat` es distinto. En la Tabla 10.7 se muestran los posibles valores asociados a estos dos sufijos. En esta tabla se aclara entre paréntesis cuál es la naturaleza de los problemas asociados a cada valor.

Los valores que indican que la solución es óptima son los siguientes. El valor 1 indica solución óptima si el problema resuelto es de programación lineal o programación lineal relajada entera-mixta. Para problemas no lineales y problemas enteros-mixtos este valor es 2 y 8, respectivamente, en la solución óptima.

El código 3 se utiliza para indicar que los modelos `lp`, `rmip`, o `nlp` no están acotados. Si ocurre esto, el usuario GAMS debe buscar en el fichero de salida la secuencia `UNBND` y determinar a qué variables afecta. Posteriormente, se debe modificar el fichero de entrada con nuevos límites sobre las variables afectadas.

Si los modelos son infactibles, el valor que devuelve el sufijo `modelstat` puede ser: 4 para modelos `lp` y `rmip`, 5 para modelos `nlp`, y 10 para modelos `mip`. En cualquiera de estos tres casos, el usuario debe buscar en el fichero de salida la secuencia `INFES` que aparece junto a las restricciones que no se han podido cumplir o a las variables cuyos valores no están comprendidos dentro de sus

límites. En estos casos, es necesario modificar los datos de entrada del problema o la estructura del problema hasta conseguir factibilidad en su solución.

Los valores anteriores (1, 2, 4, 5, 8 y 10) se obtienen al consultar la solución mediante el sufijo `modelstat`. Todos ellos se corresponden con el valor 1 para una consulta mediante el sufijo `modelstat`. Este valor indica que el optimizador no ha encontrado dificultades para obtener la solución o para determinar que ésta no existe. No obstante, si el problema es complejo y el optimizador necesita muchas iteraciones, el valor 2 indicará que el optimizador ha parado tras alcanzar el máximo permitido de iteraciones. En este caso, el usuario GAMS puede incrementar el límite máximo por defecto (1000 iteraciones) escribiendo el siguiente código antes de la aparición del comando de resolución `solve`.

```
Option iterlim=1e8;
```

En el caso de que el optimizador se detenga debido a los recursos limitados de la máquina, el valor obtenido con el sufijo `modelstat` es 3. Si 1000 segundos (valor por defecto) no son suficientes para resolver el problema se debe escribir el siguiente código antes de la aparición del comando de resolución `solve`.

```
Option reslim=1e10;
```

Por otro lado, se puede evitar que el optimizador alcance los límites anteriores cuando se resuelven modelos `mip` de gran dimensión. Para estos modelos, es posible elegir el grado de precisión de la solución entera en el óptimo. En el siguiente código aparecen las dos posibilidades que modifican dicha precisión (en ellas se muestran los valores que el optimizador entero toma por defecto):

```
Option optcr=0.1; ** O BIEN **
Option optca=0;
```

Los comandos anteriores establecen el criterio de parada que tiene en cuenta el correspondiente optimizador para modelos lineales enteros-mixtos. La primera de ellas, `optcr`, es un criterio para valores relativos, mientras que la segunda es un criterio para valores absolutos. El optimizador tomará el valor especificado por estos comandos para establecer una comparación entre cualquier solución entera y la mejor obtenida hasta entonces. Así, el valor 0.1 del primer criterio de parada indica que el optimizador se detiene cuando encuentra una solución entera cuya diferencia con la mejor solución posible sea menor que un 0.1%. En el segundo caso el optimizador se detiene cuando encuentra la solución óptima de manera exacta.

Si al consultar el estado del optimizador mediante el sufijo `solvestat` se obtiene un 4, indica que el optimizador no ha sido capaz de continuar con la resolución del problema. La razón por la que el optimizador para debe buscarse en el fichero de salida GAMS (fichero de extensión .lst).

Los valores 2, 3, y 4 para el sufijo `solvestat` se corresponden con los valores 6, 7, y 9 para el sufijo `modelstat`. Todos ellos indican que el optimizador ha alcanzado una solución intermedia (no óptima, pero factible) del problema. Los valores 6 y 7 están asociados a problemas de programación no lineal, pero se

diferencian en la "calidad" de la solución obtenida. El valor 6 indica que la solución no es factible, mientras que el valor 7 que el optimizador podría encontrar una solución factible si continuara la ejecución. Análogamente, el valor 9 indica que el optimizador no ha sido capaz de encontrar una solución factible para problemas de programación lineal entera-mixta en el tiempo disponible de ejecución.

El valor 5 para el sufijo `solvestat` indica que el optimizador necesita muchas evaluaciones de los términos no lineales en problemas de programación no lineal. En el fichero de salida se puede encontrar la ayuda necesaria para resolver este caso. Por último, los valores desde el 6 en adelante para el sufijo `solvestat` indican que el optimizador tiene grandes dificultades para resolver el problema. Estas dificultades pueden deberse a errores de preprocesado, fallos de iniciación, fallos del optimizador, errores internos del optimizador, errores de postprocesado, fallos del sistema o, simplemente, a errores no identificados.

10.3.9 La potencia del asterisco

En este apartado, se recopilan los distintos usos del asterisco en GAMS. Este símbolo puede usarse para:

- Hacer comentarios aclaratorios en el fichero de entrada. Aquellas líneas que contienen este símbolo en la primera columna son comentarios en GAMS.
- Definir conjuntos de índices de manera compacta. Esta función se explicó en el apartado 10.3.1,
- Indicar errores de compilación en el fichero de salida. Con cuatro asteriscos consecutivos se indica la línea que contiene errores.
- Indicar en el fichero de salida que las restricciones no lineales son infactibles para el punto inicial. En este caso, aparecen tres asteriscos al final de la formulación de la restricción afectada.
- Operador del producto "*" y de la potenciación "**".
- Representar un conjunto comodín cuando se definen vectores, matrices, tablas, variables o restricciones. Este comodín tiene sentido si los índices que recorren dichas estructuras no guardan relación entre sí y, por tanto, no tiene sentido agruparlos en un conjunto. El siguiente ejemplo ayuda a comprender esta característica de GAMS.

```
Set A conjunto de objetos /a1,a2/;

Table g(A,*)  características de los objetos
              alto       ancho      peso
*             (cm)       (cm)       (kg)
      a1      1.0        2.7        3.5
      a2      2.4        1.8        4.4;
```

El código anterior puede reemplazarse por el siguiente:

```
Sets A conjunto de objetos /a1,a2/
     B conjunto de características /altura, anchura, peso/;

Table g(A,B)  características de los objetos
              alto       ancho      peso
*             (cm)       (cm)       (kg)
      a1      1.0        2.7        3.5
      a2      2.4        1.8        4.4;
```

10.3.10 Resultados

En el fichero de salida GAMS (fichero con la extensión ".lst") se puede mostrar información adicional a la que aparece por defecto gracias al comando `display`, por ejemplo

```
Display x.l;
```

Para el problema del `transporte`, este comando genera la siguiente información en el fichero de salida GAMS

```
----     46 VARIABLE  X.L
cantidad de producto transportada entre i y j en toneladas
           m1          m2          m3
p1     100.000
p2                 200.000     300.000
```

La tabla anterior muestra los valores óptimos del vector x_{ij}. El sufijo `.l` sólo es válido para las variables de optimización. La salida que produce este comando contiene espacios en blanco en las posiciones de los valores por defecto, que en este caso son ceros.

Con el comando `display` se puede conocer el tiempo de cálculo empleado por el optimizador en resolver el problema. Así, la siguiente sentencia GAMS permite conocer el tiempo que transcurre al resolver el problema del `transporte`.

```
Display  transporte.resusd;
```

Esta sentencia sólo es válida después de una sentencia `solve`.

Existe otro comando similar a `display` que permite mostrar los resultados del problema en ficheros de salida alternativos y con el formato que el usuario prefiera. Este comando se explica en el apartado 10.3.14.

10.3.11 Expresiones condicionales

El símbolo"`$`" se puede utilizar en GAMS para manejar subconjuntos cuyos elementos cumplan una determinada condición. La sentencia GAMS

```
demanda(j)$(ord(j) gt 1) ..   sum(i, x(i,j))  =g=  b(j);
```

Tabla 10.8: Ejemplos que muestran la correspondencia entre el operador "$" y la sentencia "if-then-else"

expresión $	expresión if-then-else
`c('p2','m1')$(z.l ge 70)=0.01;`	`If( z.l ge 70, c('p2','m1')=0.01; );`
`c('p2','m1')=0.01$(z.l ge 70);`	`If( z.l ge 70, c('p2','m1')=0.01; else c('p2','m1')=0.0; );`

equivale a

```
demanda2..   sum(i, x(i,'m2'))  =g=  b('m2');
demanda3..   sum(i, x(i,'m3'))  =g=  b('m3');
```

La condición `$(ord(j) gt 1)` establece que existen tantas restricciones de demanda como elementos en el conjunto `j` cuyo ordinal sea mayor que 1. El funcionamiento del operador `ord` se describió en el apartado 10.3.1.

Análogamente, se puede condicionar una suma de variables o la aparición de una variable en una restricción. El siguiente código ilustra estos casos:

```
maximo(i) ..   sum(j$(ord(j) lt card(j)), x(i,j))  =l=
               b(i)$(ord(i) eq 1);
```

y es equivalente a:

```
maximo1..  x('p1','m1') + x('p1','m2') =l= b('p1');
maximo2..  x('p2','m1') + x('p2','m2') =l= 0 ;
```

Las expresiones condicionales típicamente se emplean en la asignación de datos o en las sentencias de escritura de datos de salida (véase el apartado 10.3.14). En ambos casos, el operador "$" puede reemplazarse por una sentencia "if-then-else". En la Tabla 10.8 se muestra un ejemplo de asignación de datos condicionada. El lector debe asegurarse dónde colocar el símbolo '$', pues dependiendo del lugar el resultado de la operación es distinto.

En los apartados 12.1, 12.2.1, y 12.7 se presentan algunos ejemplos adicionales que hacen uso de sentencias condicionales.

En las expresiones condicionales de GAMS se pueden utilizar los operadores típicos de otros lenguajes de programación, como son

- not, and, or, xor como operadores lógicos;
- < (lt), <= (le), = (eq),<> (ne), >= (ge), > (gt) como operadores relacionales.

Tabla 10.9: Ejemplos con conjuntos dinámicos

Expresión matemática	Expresión en GAMS
$k = \{a, b, c\}$ $s \subseteq k$	`Set k /a,b,c/` `s(k);`
$s = \{a, b, c\}$	s(k)=yes;
$s = \emptyset$	`s(k)=no;`
$s = \{c\}$	`s(k)=no;` `s(k)$(ord(k) eq card(k))=yes;`
$s = \{b, c\}$	`s(k)=no;` `s(k)$(ord(k) gt 1)=yes;`

10.3.12 Conjuntos dinámicos

Los conjuntos dinámicos son una de las estructuras más útiles que ofrece GAMS. Los elementos de estos conjuntos pueden variar a lo largo de la ejecución del programa. Estos conjuntos siempre se definen como subconjuntos de conjuntos constantes o estáticos. La Tabla 10.9 muestra la equivalencia entre expresiones matemáticas y expresiones GAMS que usan conjuntos dinámicos.

En la primera fila de la Tabla 10.9, se define el conjunto estático `k` y en la segunda su subconjunto dinámico `s(k)`. Para poder utilizar los conjuntos dinámicos es necesario establecer su estado inicial. En la tercera fila se asigna el conjunto vacío como estado inicial. En la cuarta fila se asigna el último elemento del conjunto `k` al subconjunto `s(k)`. Esta asignación hace uso de los operadores `ord(k)` y `card(k)`. El operador '`card`' obtiene el número de elementos de un conjunto estático o dinámico, y el operador `ord` obtiene la posición de un elemento en un conjunto estático (este último operador sólo es válido para conjuntos estáticos). La última fila muestra cómo asignar dos elementos (b y c) al subconjunto dinámico `s(k)`.

Los conjuntos dinámicos también son útiles para operar con conjuntos. Los ejemplos de la Tabla 10.10 muestran las operaciones típicas con conjuntos.

Otros ejemplos donde se emplean conjuntos dinámicos aparecen en el apartado 12.2.1.

10.3.13 Estructuras iterativas

Los bucles, por ejemplo, permiten resolver colecciones de problemas similares que requieren una resolución iterativa debido al empleo de técnicas de descomposición. El código siguiente, basado en el ejemplo 4.1, combina los bucles y los conjuntos dinámicos para resolver un problema iterativamente.

Tabla 10.10: Ejemplos de operaciones con conjuntos

Expresión matemática	Expresión en GAMS
$A = \{a1, a2, a3, a4, a5\}$	`set A conjunto estatico /a1*a5/;`
$b \subseteq A$, $b = \{a1, a2, a5\}$	`set B(A) subconjunto /a1,a2,a5/;`
$c \subseteq A$, $c = \{a2, a3, a5\}$	`set C(A) subconjunto /a2,a3,a5/;`
$b \cup c = \{a1, a2, a3, a5\}$	`set UN(A) subconjunto dinámico;` `UN(A)=B(A)+C(A);`
$b \cap c = \{a2, a5\}$	`set IN(A) subconjunto dinámico;` `IN(A)=B(A)*C(A);`
$\bar{b} = \{a3, a4\}$	`set COMP(A) subconjunto dinámico;` `COMP(A)=not B(A);`
$b - c = \{a1\}$	`set DIFF(A) subconjunto dinámico;` `DIFF(A)=B(A)-C(A);`

```
Sets K /1*3/
     DIN(K);

Parameter m(K) pendiente de C2; m('1')=1;
Positive variables x1,x2;
Free variable z; x1.up=3;

Equations
 Cost
 C1
 C2(K)
 C4
 C6;

Cost.. z=e=3*x1+x2; C1.. -x1+x2=l=2; C2(DIN).. x1+m(DIN)*x2=l=6;
           C4.. 2*x1-x2=l=4; C6.. -x1-x2=l=-1;

Model ejcap4 /all/;

loop(K, DIN(K)=yes; Solve ejcap4 using lp maximizing z; m(K+1)=m(K)+1;
);
```

El bucle anterior permite resolver tres problemas similares secuencialmente. La resolución de estos problemas puede ser de utilidad para una análisis de sensibilidad sobre la restricción activa `C2` (en cada iteración se añade una restricción que cambia la pendiente de esta recta). En tiempo de ejecución las siguientes restricciones no varían:

```
C1.. - x1 + x2 =l=  2 ;
C4.. 2*x1 - x2 =l=  4 ;
C6.. - x1 - x2 =l= -1 ;
```

Tabla 10.11: Formato de las sentencias `for` y `while`

```
scalar contite contador de iteraciones;
for(contite= 1 to 100,
 ...
 ...
);
```

```
scalar contite; contite=1;
while(contite lt 100,
 ...
 contite=contite+1;
);
```

Para la primera iteración (`K = 1`), el valor de la función objetivo es 12 (ésta es la solución del problema base) donde la restricción `C2` tiene la forma:

```
C2(1)..   x1 + x2 =l=  6 ;
```

En la segunda iteración (`K = 2`), el valor de la función objetivo es 10 y la restricción `C2` se modela como

```
C2(1)..   x1 + x2 =l=  6 ;
C2(2)..   x1 + 2*x2 =l=  6 ;
```

En la última iteración (`K = 3`), el valor de la función objetivo es 8.9 y la restricción `C2` se toma como

```
C2(1)..   x1 + x2 =l=  6 ;
C2(2)..   x1 + 2*x2 =l=  6 ;
C2(3)..   x1 + 3*x2 =l=  6 ;
```

Nótese que las restricciones `C2` que se han escrito anteriormente no pueden usarse como código en el fichero de entrada GAMS.

En GAMS también se emplean otras estructuras iterativas típicas de los lenguajes de programación, como son `for` y `while`. Estas dos estructuras son de utilidad si el número de iteraciones depende de un escalar y no de un conjunto (véase la Tabla 10.11). La sentencia `while` se usa normalmente cuando el criterio de parada del bucle depende de una expresión condicional. Por otro lado, la sentencia `for` se suele emplear si el número de iteraciones del bucle se conoce con antelación y no depende de ninguna condición.

A continuación se muestra cómo emplear una sentencia `for` en el problema del transporte.

```
...
scalar contite "contador de iteraciones";

for(contite= 1 to 3,
 Solve transporte using lp minimizing z ;
 b(j)=b(j)+ord(j);
);
```

El código equivalente al anterior empleando la sentencia `while` es:

```
...
scalar contite; contite=1;

while(contite le 3,
 Solve transporte using lp minimizing z ;
 b(j)=b(j)+ord(j); contite=contite+1;
);
```

Con cualquiera de los dos códigos se resuelven tres problemas secuencialmente. En la primera iteración (`contite = 1`) GAMS tiene en cuenta el siguiente grupo de restricciones de `demanda`

```
demanda(m1)..  x(p1,m1) + x(p2,m1) =G= 100 ;
demanda(m2)..  x(p1,m2) + x(p2,m2) =G= 200 ;
demanda(m3)..  x(p1,m3) + x(p2,m3) =G= 300 ;
```

Para la segunda iteración (`contite = 2`) el término independiente de las anteriores restricciones se modifica y pasa de ser 100, 200 y 300 a ser 101, 202, y 303, respectivamente. Y, finalmente, en la tercera iteración (`itecount = 3`) los términos independientes son 102, 204, y 306.

10.3.14 Escritura en fichero de salida

El comando `put` permite decidir a los usuarios de GAMS qué ficheros de salida crear, qué información escribir y el formato de escritura. Puede resultar útil para agrupar los resultados en distintos ficheros a conveniencia del usuario. A pesar de la mayor flexibilidad en el formato de escritura de datos, este comando es un poco más complejo de utilizar que el comando `display`. Por ejemplo, para producir la misma salida que el comando `display` en el problema del transporte, se necesitaría el siguiente código:

```
file out /transporte.out/;
put out;
put "----     46 VARIABLE  X.L  cantidad de producto transportada
              entre i y j en toneladas"//;
put "            ";
loop(j,put j.tl:12);
put //;
loop(i,
     put i.tl:2;
     loop(j,
          put$(x.l(i,j) ne 0)  x.l(i,j):12:3;
          put$(x.l(i,j) eq 0) "            ";
     );
     put /;
);
```

En la primera línea del código anterior, se especifica el nombre del fichero de salida mediante el comando `file`. El identificador `out` lo establece el usuario y se refiere al nombre del fichero interno de GAMS que se asocia al fichero de

salida (en este caso, `transporte.out`). A cada fichero de salida debe asignarsele un etiqueta diferente.

Mientras que el comando `display` trata automáticamente los índices de las variables o datos, el comando `put` debe recorrer estos índices mediante estructuras iterativas.

Como se puede observar en el ejemplo anterior, el comando put puede ir seguido de:

- El identificador interno del fichero de salida (véase la segunda línea). Tras esta sentencia, todo se escribirá en el fichero de salida asociado. Si se quiere escribir en un nuevo fichero se debe utilizar un identificador diferente.
- Texto entre comillas. El usuario escribe el texto tal y como quiere que aparezca en el fichero de salida. Ésto se hace en las líneas 3, 4, y 11 del código anterior.
- El símbolo "`/`". Este símbolo se usa para introducir un retorno de carro en el fichero de salida (véanse las líneas 3, 6, y 13).
- Identificadores de conjuntos, vectores y/o variables seguidos de los sufijos correspondientes. El sufijo `tl` se usa para escribir el identificador de un conjunto, como en la línea 5. En la línea 11 se utiliza el sufijo `l` para escribir el valor de la variable de optimización.
- Una expresión condicional. Si la escritura de los datos depende de que se cumpla una determinada condición, se utiliza el comando "`$`" (véanse líneas 10 y 11) o la sentencia `if-then-else`.

10.3.15 Listado de las restricciones no lineales

El fichero de salida que genera GAMS automáticamente proporciona cierta información que ayuda al usuario a depurar el programa y detectar errores. No son necesarias muchas nociones de lengua inglesa para entender lo que aparece en este fichero. Sin embargo, cuando se trata de comprobar si las restricciones no lineales están bien formuladas, el fichero de salida no es lo suficientemente autoexplicativo.

El siguiente código representa el problema de minimizar la función $e^{-x_1}x_2^3$, considerando como puntos iniciales $x_1^0 = 1$ y $x_2^0 = 0.5$:

```
 POSITIVE VARIABLE X1,X2; FREE VARIABLE z;
 X1.L=1; X2.L=0.5;

EQUATION
 COSTE    esta es la funcion objetivo;

COSTE..   exp(-X1)*power(X2,3)=e=z;

MODEL nolineal /all/; SOLVE nolineal USING nlp MINIMIZING z;
```

La parte correspondiente a las restricciones no lineales en el fichero de salida es

```
COSTE..- (0.046)*X1+(0.2759)*X2-Z =E= 0; (LHS = 0.046, INFES = 0.046 ***)
```

En la expresión anterior, los coeficientes entre paréntesis representan la evaluación de la primera derivada de la restricción `COSTE` respecto a cada variable en el punto inicial; es decir, $-e^{-x_1^0}\cdot(x_2^0)^3 = -0.046$ y $e^{-x_1^0}\cdot 3\cdot(x_2^0)^2 = 0.2759$. Además, se observan dos igualdades después del punto y coma. La primera igualdad es la evaluación del término dependiente (LHS) de la restricción `COSTE` para el punto inicial: $e^{-x_1^0}\cdot(x_2^0)^3 = 0.046$. La segunda indica que esta restricción es infactible en el punto inicial, aunque no significa que sea infactible en cualquier punto.