Optimización con LINGO

Carlos Ivorra

1 Introducción

LINGO es una aplicación capaz de resolver modelos de programación matemática. Estas notas se han elaborado considerando la versión 12.0. En esta primera sección describiremos mínimamente el uso de la aplicación en sí, mientras que en las siguientes describiremos el lenguaje que hemos de emplear para introducir los modelos en LINGO y la interpretación de las soluciones que éste proporciona.

Una vez instalado, LINGO trabaja con dos clases de documentos: documentos con extensión .lg4, en los que escribimos el modelo que queremos resolver (y se crean desde el menú File → New) y documentos con extensión .lgr, en los que LINGO escribe la solución. Ambos se pueden guardar de la forma habitual, con los menús File → Save y File → Save as...

Una diferencia es que, si al salir de LINGO hay algún documento .lg4 sin guardar, aparece un cuadro de diálogo que nos pregunta si queremos guardarlo, mientras que los documentos .lgr se pierden si no se guardan.

Por otro lado, una vez que LINGO ha creado un documento .lgr con datos sobre un modelo, podemos modificarlo a nuestro gusto, borrando cualquier información que no nos interese, añadiendo cualquier clase de explicaciones, títulos, comentarios, etc., pegando en un único documento el contenido de varios documentos .lgr o incluso .lg4, etc.

Por el contrario, lo que escribamos en un documento .lg4 debe ser correcto en el lenguaje de LINGO, pues en otro caso al intentar resolver el problema obtendremos mensajes de error en lugar de la solución deseada.

No vamos a describir aquí las funciones de los menús e iconos de LINGO, si bien muchos de ellos realizan tareas típicas que podemos encontrar en muchas otras aplicaciones. Destacamos únicamente el menú Edit → Paste function, que despliega varios de los comandos y funciones disponibles en el lenguaje de LINGO, y el menú LINGO → Options..., que nos permite configurar muchas características del funcionamiento de LINGO. De momento destacamos las siguientes:

- General Solver
  - Dual computations: Podemos seleccionar que LINGO calcule los multiplicadores de Kuhn y Tucker o variables duales (Prices), o también los intervalos de sensibilidad de los coeficientes de la función objetivo (en programación lineal) y de los términos independientes de las restricciones (Prices & Ranges) o nada (None). Hay una cuarta opción que limita el cálculo de los multiplicadores para el caso en que éste consuma demasiado tiempo.

    Si la opción Prices & Ranges está seleccionada, LINGO puede presentar mensajes de error con algunos problemas de programación no lineal (incluso en algunos muy sencillos).
  - Variables assumed non-negative: Con esta opción LINGO supone que todas las variables de un problema son ≥ 0 salvo que se indique explícitamente lo contrario. En los ejemplos de estas notas supondremos siempre que esta opción está seleccionada.
- Global Solver
  - Use global solver: Esta opción hace que LINGO busque óptimos globales en lugar de óptimos locales. En problemas grandes puede ralentizar mucho el proceso de resolución, pero en

problemas pequeños mejora el resultado en muchas ocasiones. Naturalmente, sólo es relevante en problemas no lineales.

- Model Generator
  - Unary Minus Priority: Es conveniente fijar esta opción en Low. Su efecto es que, por ejemplo, LINGO interprete -2^2 como -4 en lugar de como 4.
  - Assume model is linear: Esta opción optimiza el precesamiento del modelo para problemas lineales.

2 Introducción básica de un modelo en LINGO

Veamos cómo introducir en LINGO el problema siguiente:

Una empresa elabora tres tipos de piensos usando cuatro tipos de cereales. Cada saco de pienso contiene 50 *kg. y se vende al precio (en euros) indicado en la tabla siguiente, que contiene también la composición de cada saco y las existencias de cereales en la fábrica:*

Pienso	*Avena*	*Maíz*	*Cebada*	*Mijo*	*Precio*
1	25	25	0	0	9
2	0	20	20	10	12
3	20	0	30	0	6.20
Existencias	50 000	80 000	40 000	10 000	

Determina el número de sacos que deberá producir la empresa de cada tipo de pienso para maximizar el ingreso (supuesto que vende toda su producción).

El modelo matemático correspondiente a este problema es:

$$\begin{aligned} \text{Max.} \quad & 9x + 12y + 6.2z \\ \text{s.a} \quad & 25x + 20z \leq 50\,000 \\ & 25x + 20y \leq 80\,000 \\ & 20y + 30z \leq 40\,000 \\ & 10y \leq 10\,000 \\ & x,\, y,\, z \geq 0 \end{aligned}$$

Y la forma de escribirlo en LINGO (en un documento .lg4) es la siguiente:

```
[Ingresos] Max = 9*x+12*y+6.2*z;
[Avena]  25*x+20*z < 50000;
[Maiz]   25*x+20*y < 80000;
[Cebada] 20*y+30*z < 40000;
[Mijo]   10*y < 10000;
```

Observaciones:

- La función objetivo se introduce precedida de `Max =` o de `Min =`.
- Para introducir una desigualdad de $\leq$ o $\geq$ se escribre `<=` o `>=` aunque se puede abreviar a `<` o `>`. Para introducir una igualdad escribimos simplemente `=`
- Si hemos seleccionado la opción Variables assumed non-negative no es necesario introducir las condiciones de signo.
- Cada instrucción termina obligatoriamente con `;`

- Los cambios de línea son irrelevantes. Por claridad podemos escribir cada ecuación en una línea, pero sería equivalente escribirlas todas seguidas en la misma línea. Recíprocamente, si una restricción es muy larga, podemos partirla en dos o más líneas (teniendo en cuenta que un cambio de línea es como un espacio en blanco, por lo que no podemos partir una palabra o un número).

- Las etiquetas entre corchetes verb/[ ]/ son opcionales. Sirven únicamente para relacionar más fácilmente la solución que proporciona LINGO con las distintas líneas del modelo. En caso de introducir etiquetas, éstas no pueden contener espacios en blanco ni signos especiales, como acentos, eñes, etc. Una etiqueta puede contener números, pero no empezar por un número. Podemos usar el guión bajo para separar palabras (p.ej.: `[Existencias_de_mijo]`).

- Las mismas consideraciones que acabamos de hacer para las etiquetas valen para los nombres de las variables. En lugar de `x, y, z` podríamos haber elegido `x1, x2, x3`, o incluso `sacos_1`, `sacos_2`, `sacos_3`.

- LINGO no distingue entre mayúsculas o minúsculas, de modo que si en un lugar escribimos `x` y en otro `X`, para LINGO se tratará de la misma variable. Del mismo modo, da igual escribir `Max`, `MAX` o `max`.

- No se pueden omitir los signos de multiplicación `*`. Para introducir, por ejemplo, $(x-5)^3$, escribiríamos `(x-5)^3`. Las funciones matemáticas disponibles en LINGO (exponenciales, logaritmos, etc.) pueden consultarse en el menú Edit → Paste function → Mathematical / Trigonometrical

- Es posible insertar líneas de comentarios (es decir, líneas que LINGO no leerá) sin más que empezarlas por un signo `!` (y terminarlas igualmente con `;`).

3 Interpretación de la solución

Una vez introducido un modelo, LINGO lo resuelve si seleccionamos el menú LINGO → Solve (o, equivalentemente, pulsando en el icono en forma de diana). LINGO genera entonces un documento .lgr que, en el caso del ejemplo anterior (además de alguna información técnica) contiene lo siguiente:

Variable	Value	Reduced Cost
X	2000.000	0.000000
Y	1000.000	0.000000
Z	0.000000	1.000000

Row	Slack or Surplus	Dual Price
INGRESOS	30000.00	1.000000
AVENA	0.000000	0.3600000
MAIZ	10000.00	0.000000
CEBADA	20000.00	0.000000
MIJO	0.000000	1.200000

- La columna titulada `Value` contiene el valor óptimo de cada variable.

- En la columna titulada `Slack or Surplus`, la entrada correspondiente a la función objetivo (es decir, la etiquetada como `INGRESOS`) contiene el valor óptimo de la función objetivo, mientras que las restantes contienen las variables de holgura de las restricciones. Por ejemplo, vemos que para la solución óptima sobran 10 000 kg de maíz.

- La columna titulada `Reduced Cost` contiene (salvo el signo) los multiplicadores de Kuhn y Tucker de las variables del problema. El signo es el que se corresponde con la interpretación siguiente:

El coste reducido de una variable x indica aproximadamente lo que empeorará la función objetivo (es decir, disminuirá en un problema de maximizar o aumentará en un problema de minimizar) por cada unidad que aumente el término independiente de la restricción $x \geq 0$.

Estrictamente hablando, el coste reducido λ ha de entenderse como la derivada de la función valor óptimo respecto al término independiente indicado[1] (cambiada de signo en los problemas de maximizar), por lo que la mejora de la función objetivo cuando la condición de signo pasa a ser $x \geq c$ será aproximadamente $\lambda \cdot c$, y ésta aproximación sólo es válida para valores de c suficientemente pequeños.

Para problemas lineales el coste reducido tiene una interpretación[2] alternativa:

El coste reducido de una variable x que tome el valor 0 es lo que debe mejorar el coeficiente de x en la función objetivo para que el valor óptimo de x pase a ser no nulo. (Las variables que ya son no nulas tienen coste reducido nulo.)

Así, por ejemplo, por cada saco de pienso de tipo 3 que quisiéramos fabricar, los ingresos disminuirían en 1€ o, equivalentemente, para que resulte rentable fabricar sacos de pienso de tipo 3 es necesario que su precio de venta aumente al menos en 1€.

- La columna `Dual Price` contiene (salvo el signo) los multiplicadores de Kuhn y Tucker (o variables duales, en el caso de la programación lineal) de las restricciones (excepto la entrada correspondiente a la función objetivo, cuyo valor es siempre igual a 1). El signo es el que se corresponde con la interpretación siguiente:

  El precio dual de una restricción indica aproximadamente lo que mejorará la función objetivo por cada unidad que aumente el término independiente de la restricción.

  Esto ha de entenerse igualmente como una derivada en las mismas condiciones en que es válida la interpretación de los costes reducidos.

  Por ejemplo, por cada kg adicional de avena que pudiéramos conseguir los ingresos aumentarían en 0.36€.

Si hemos seleccionado la opción para que LINGO calcule los intervalos de sensibilidad, podemos obtenerlos en el menú LINGO → Range (teniendo activa la ventana correspondiente al documento .lg4). El resultado es un nuevo documento .lgr con las tablas siguientes (que en la práctica podemos copiar y pegar en la ventana que contiene la solución del problema):

```
                    Objective Coefficient Ranges:

                         Current        Allowable        Allowable
Variable             Coefficient         Increase         Decrease
       X            9.000000          INFINITY         1.250000
       Y            12.00000          INFINITY         12.00000
       Z            6.200000         1.000000          INFINITY
```

[1] Esto bajo el supuesto de que la solución óptima sea un punto de Kuhn y Tucker y que la función valor óptimo sea diferenciable. En caso contrario la interpretación anterior no es válida. Esto sucede si y sólo si la solución óptima es un punto regular del problema, es decir, si los gradientes de las restricciones saturadas son linealmente independientes.

[2] La condición necesaria y suficiente para que cualquiera de las dos interpretaciones sea válida en el caso de la programación lineal es que la solución óptima no sea degenerada, es decir, que no tenga ninguna variable básica nula.

```
                    Righthand Side Ranges:

                        Current       Allowable       Allowable
Row                         RHS        Increase        Decrease
AVENA                  50000.00        10000.00        50000.00
MAIZ                   80000.00        INFINITY        10000.00
CEBADA                 40000.00        INFINITY        20000.00
MIJO                   10000.00         5000.000       10000.00
```

En problemas de programación lineal, la interpretación es la siguiente:[3]

- La tabla titulada `Objective Coefficient Ranges` contiene el valor actual del coeficiente de cada variable en la función objetivo junto con lo máximo que puede aumentar o disminuir para que la solución óptima no cambie.

  Por ejemplo, mientras el precio de los sacos de tipo 1 no descienda más de 1.25€, la solución óptima del problema seguirá siendo la misma.

- La tabla titulada `Righthand Side Ranges` contiene el valor actual de término independiente de cada restricción junto con lo máximo que puede aumentar o disminuir para que las variables básicas de la solución óptima sigan siendo las mismas. Si la restricción no está saturada podemos decir que la solución óptima seguirá siendo la misma.

  Por ejemplo, mientras la cantidad disponible de avena no aumente en más de 10 000 kg seguirá siendo cierto que

  1. No convendrá producir pienso de tipo 3 ($z = 0$),
  2. Agotaremos toda la avena disponible (la holgura de la avena será nula),
  3. Agotaremos todo el mijo disponible (la holgura del mijo será nula).

  Se cumple que el precio dual de una restricción sólo es aplicable para determinar (de forma exacta en programación lineal) la mejora de la función objetivo que tiene lugar cuando se produce un incremento en el término independiente de la restricción que queda dentro del rango indicado en la tabla que acabamos de interpretar.

Terminamos con algunas observaciones:

- Si intentamos resolver un problema infactible o no acotado, LINGO lo indica mediante un cuadro de diálogo.
- Si "resolvemos" un problema sin función objetivo (por ejemplo, poniendo una exclamación ante la primera línea de nuestro modelo de ejemplo), LINGO estudia si las restricciones introducidas son o no factibles.
- Si no hemos puesto etiquetas a las restricciones, LINGO se referirá a ellas con números (según el orden en que las hemos escrito en el modelo), de ahí la conveniencia de poner etiquetas.

4 Restricciones sobre los dominios de las variables

- La instrucción `@BND(1000,x,1900);` hace que la variable `x` pueda variar entre 1 000 y 1 900, es decir, que tiene el mismo efecto que incorporar las restricciones `x>1000; x<1900;`

  Si añadimos esta instrucción al ejemplo de las secciones anteriores, la nueva salida de LINGO contiene la línea

[3] Esta interpretación sólo es válida en el caso en que la solución óptima no es degenerada, es decir, que ninguna variable básica es nula.

```
                    Variable           Value        Reduced Cost
                           X        1900.000           -1.250000
```

Aquí hay que entender que el coste reducido es (salvo el signo) el multiplicador de Kuhn y Tucker de la restricción $x \leq 1900$, y nos indica que por cada unidad que aumentáramos el número máximo de sacos de pienso permitidos de tipo 1 los ingresos empeorarían en -1.25€, es decir, aumentarían en 1.25€. En general, el coste reducido se refiere a la cota (inferior o superior) que esté saturada (y es nulo si ninguna lo está).

No obstante, una instrucción `@BND` no equivale necesariamente a dos restricciones. Por ejemplo, si añadimos las restricciones `x>-4; x<10;` teniendo en cuenta que suponemos marcada la opción `Variables assumed non-negative`, el rango de variación de x es en realidad $0 \leq x \leq 10$. Sin embargo, si escribimos `@BND(-4,x,10);` con ello cancelamos la condición de no negatividad, y permitimos que la variable `x` varíe entre -4 y 10.

Más claramente: LINGO asocia a cada variable una cota inferior y una cota superior. Cuando se selecciona la opción de variables no negativas la cota inferior de cada variable es 0 por defecto, y la cota superior es un número muy grande por defecto. La instrucción `@BND` modifica el valor de estas cotas.

- La instrucción `@Free(x);` declara la variable `x` como variable libre, es decir, establece a cota inferior de la variable en un número muy pequeño (muy negativo) y la cota superior en un número muy grande. En particular, cancela para la variable en cuestión la condición de no negatividad, si es que ésta está establecida por defecto.
- La instrucción `@GIN(x);` obliga a la variable `x` a tomar valores enteros (positivos o negativos, salvo que las variables sean no negativas por defecto).
- La instrucción `@BIN(x);` obliga a la variable `x` a tomar únicamente los valores 0 o 1.
- La instrucción `@Semic(10,x,20);` convierte a `x` en una variable semicontinua, que ha de tomar el valor 0 o bien estar en el rango $10 \leq x \leq 20$.

La introducción en un modelo de variables enteras, binarias o semicontinuas hace que las interpretaciones indicadas en la sección anterior de los precios duales y costes reducidos dejen de ser válidas.

5 Definición de conjuntos

LINGO permite introducir los modelos en términos de conjuntos de índices, lo cual aporta numerosas ventajas:

- Las ecuaciones son independientes de los datos, de modo que se puede cambiar éstos sin modificar aquéllas, o incluso se puede escribir un modelo que contenga únicamente las ecuaciones y que lea los datos de otro documento.
- Si el modelo tiene varias ecuaciones que siguen un mismo esquema, se pueden introducir todas ellas como una única fórmula general.
- La estructura del modelo se simplifica, ya que, por una parte, las ecuaciones pueden escribirse conceptualmente, sin mezclarlas con los datos y, por otra, los datos pueden introducirse en un orden más claro, independiente del lugar en el que deben aparecer en el modelo.

Veamos cómo modelizar el ejemplo de las secciones precedentes en términos de conjuntos. Para ello observamos que todos los datos del problema están asociados esencialmente a dos conjuntos: un conjunto de cuatro cereales y un conjunto de tres tipos de pienso. Una forma de introducir estos conjuntos en LINGO es la siguiente:

```
SETS:
Cereal/Avena, Maiz, Cebada, Mijo/:Existencias;
Pienso/1..3/: Precio, Sacos;
ENDSETS
```

- Las palabras `SETS:` `ENDSETS` determinan una *sección* en un modelo de LINGO. En general, una sección empieza con su nombre (en este caso `SETS`) seguido de `:` (no `;`) y termina con `END` y el mismo nombre (sin ningún signo de puntuación). La sección `SETS` sirve para definir conjuntos.

- Para definir un conjunto escribimos su nombre, luego sus elementos entre barras `/ /` y luego, separadas por `:`, las variables asociadas al conjunto.

  Así, por ejemplo, las líneas anteriores definen un conjunto llamado `Cereal` cuyos elementos son `Avena, Maiz, Cebada, Mijo`, de modo que cada cereal tiene asociada una variable `Existencias`, es decir, que hemos definido cuatro variables[4] llamadas `Existencias(Avena)`, `Existencias(Maiz)`, `Existencias(Cebada)` y `Existencias(Mijo)`.

  Similarmente, hemos definido un conjunto llamado `Pienso` cuyos elementos son los números del 1 al 3, y estos elementos tienen definidas dos variables, `Precio` y `Sacos`, de modo que tenemos seis nuevas variables, `Precio(1)`, `Precio(2)`, `Precio(3)`, `Sacos(1)`, `Sacos(2)`, `Sacos(3)`.

  En vez de `1..3` podíamos haber escrito `1, 2, 3`, pero los dos puntos `..` son una de las formas abreviadas que admite LINGO para definir los elementos de un conjunto. En general tenemos las posibilidades siguientes (aparte de la enumeración explícita de los elementos):

Definición	**Elementos**
`4..7`	4, 5, 6, 7
`CH2..CH5`	CH2, CH3, CH4, CH5
`MON..WED`	MON, TUE, WED
`OCT..JAN`	OCN,NOV,DEC, JAN
`NOV2010..FEB2011`	NOV2010, DEC2010, JAN2011, FEB2011

Podemos ver las variables que hemos definido "resolviendo" el problema que hemos tecleado hasta ahora (que de momento sólo consta de la sección `SETS`). Entonces LINGO nos dice que el problema es factible y presenta como ejemplo de solución factible la siguiente:

```
                   Variable           Value
         EXISTENCIAS( AVENA)        1.234568
          EXISTENCIAS( MAIZ)        1.234568
        EXISTENCIAS( CEBADA)        1.234568
          EXISTENCIAS( MIJO)        1.234568
                  PRECIO( 1)        1.234568
                  PRECIO( 2)        1.234568
                  PRECIO( 3)        1.234568
                   SACOS( 1)        1.234568
                   SACOS( 2)        1.234568
                   SACOS( 3)        1.234568
```

En general, cuando busca soluciones factibles, LINGO asigna el valor `1.234568` a las variables que no están sujetas a ninguna restricción.

[4] A estas variables les vamos a asignar valores constantes, las existencias de cada cereal dadas en el enunciado del problema, pero en principio LINGO no distingue entre variables y constantes, sino que una variable se convierte en constante en el momento en que se le asigna un valor fijo.

Para asignar un valor fijo a algunas variables (y convertirlas así en constantes) usamos una sección `DATA`:

```
SETS:
Cereal/Avena, Maiz, Cebada, Mijo/:Existencias;
Pienso/1..3/: Precio, Sacos;
ENDSETS

DATA:
Existencias = 50000   80000   40000   10000;
Precio      = 9, 12, 6.20;
ENDDATA
```

Notemos que hemos separado por comas los distintos precios, mientras que no hemos puesto nada entre las existencias de cada cereal. El uso de comas es opcional (al igual que lo es en la declaración de los elementos de un conjunto en la sección `SETS`). Si ahora volvemos a "resolver" el problema veremos que las constantes ya tienen su valor asignado, y que sólo quedan las variables `Sacos`, que son las auténticas variables del problema:

```
                   Variable           Value
        EXISTENCIAS( AVENA)        50000.00
         EXISTENCIAS( MAIZ)        80000.00
       EXISTENCIAS( CEBADA)        40000.00
         EXISTENCIAS( MIJO)        10000.00
                PRECIO( 1)        9.000000
                PRECIO( 2)        12.00000
                PRECIO( 3)        6.200000
                 SACOS( 1)        1.234568
                 SACOS( 2)        1.234568
                 SACOS( 3)        1.234568
```

- Si hubiéramos querido asignar a todas las existencias el valor 10 000 podríamos haber escrito

  ```
  Esistencias = 10000;
  ```

- Si hubiéramos querido dejar las existencias de cebada como variable podríamos haber escrito:

  ```
  Existencias = 50000, 80000, , 10000;
  ```

  (Éste es el único caso en que el uso de comas es obligatorio.)

- La declaración de los elementos de un conjunto puede hacerse indistintamente en la sección `SETS` o en la sección `DATA`. Así, por ejemplo, podríamos haber escrito:

  ```
  SETS:
  Cereal:Existencias;
  Pienso/1..3/: Precio, Sacos;
  ENDSETS

  DATA:
  Cereal      = Avena   Maiz   Cebada   Mijo;
  Existencias = 50000   80000  40000    10000;
  Precio      = 9, 12, 6.20;
  ENDDATA
  ```

Igualmente podríamos haber escrito `Pienso = 1..3;` en la sección `DATA`, pero siempre cuidando de no definir nunca dos veces el mismo conjunto.

- Si quisiéramos definir una constante no asociada a ningún conjunto (por ejemplo, un máximo número de sacos que nos estuviera pemitido producir) lo haríamos en la sección `DATA`, de este modo:

  ```
  MaxSacos = 2500;
  ```

Nos falta introducir la cantidad de cada cereal que contiene cada tipo de pienso. Estas cantidades no están asociadas ni a los cereales ni a los piensos, sino a las parejas de ambos. Para ello hemos de definir el conjunto de tales parejas. La forma de hacerlo es la siguiente:

```
SETS:
Cereal:Existencias;
Pienso/1..3/: Precio, Sacos;
Pareja(Pienso, Cereal): Cantidad;
ENDSETS

DATA:
Cereal      = Avena   Maiz   Cebada   Mijo;
Cantidad    = 25      25     0        0
              0       20     20       10
              20      0      30       0;
Existencias = 50000   80000  40000    10000;
Precio      = 9, 12, 6.20;
ENDDATA
```

- La definición `Pareja(Pienso, Cereal)` establece que el conjunto `Pareja` está formado por todos los pares de un pienso y un cereal. Cada pareja tiene asociada una variable `Cantidad`, de modo que hemos definido 12 nuevas variables `Cantidad(1, Avena)`, etc. Igualmente se pueden definir conjuntos de ternas, cuádruplas, etc.

- A la hora de dar un valor a todas las variables asociadas a un conjunto de parejas, ternas, etc. se fijan todos los índices menos el último y se hace variar éste, luego se aumenta el penúltimo y se vuelve a hacer variar el último, y así sucesivamente. Por ejemplo, si definiéramos una variable `a(I, J, K)` donde cada uno de los tres conjuntos tiene elementos `1, 2`, habría que introducir los valores de a en el orden $a_{111}, a_{112}, a_{121}, a_{122}, a_{211}, a_{212}, a_{221}, a_{222}$.

- Como siempre, los valores sucesivos se pueden separar por comas o no, y tampoco hay ninguna necesidad de cambiar de línea al completar una fila de una tabla, ni mucho menos de alinear sus columnas. Si lo hemos hecho así en la tabla del ejemplo ha sido únicamente por claridad.

- También habríamos podido definir simultáneamente el conjunto `Cereal` y las variables `Existencias`, así:

  ```
  Cereal, Existencias = Avena 50000, Maiz 80000, Cebada 40000, Mijo 10000;
  ```

  (Como siempre, las comas se pueden suprimir.)

Terminamos esta sección observando que también es posible asignar un valor inicial a una variable sin que por ello LINGO deje de considerarla variable. Para ello no asignamos el valor en la sección `DATA`, sino en una sección `INIT`, así:

```
INIT:
sacos = 1000;
ENDINIT
```

Con esto las tres variables `sacos` toman el valor inicial 1 000. Si sólo quisiéramos asignar dicho valor a `sacos(1)` tendríamos que haber escrito `sacos = 1000, , ;`

Esto puede ser útil cuando conocemos una solución próxima a la solución óptima (por ejemplo, porque vamos a resolver un problema que resulta de modificar levemente otro problema cuya solución óptima conocemos) o bien cuando LINGO tiene dificultad en encontrar una solución factible y nosotros conocemos una. En situaciones de este tipo, proporcionar a LINGO una buena solución inicial puede ayudarle a encontrar más rápidamente la solución óptima.

6 Ecuaciones con conjuntos

Veamos ahora cómo introducir las ecuaciones de un modelo cuando los datos están expresados en términos de conjuntos. Para el ejemplo que estamos considerando tendríamos que escribir:

```
SETS:
Cereal:Existencias;
Pienso/1..3/: Precio, Sacos;
Pareja(Pienso, Cereal): Cantidad;
ENDSETS

DATA:
Cereal      = Avena   Maiz   Cebada   Mijo;
Cantidad    = 25      25     0        0
              0       20     20       10
              20      0      30       0;
Existencias = 50000   80000  40000    10000;
Precio      = 9, 12, 6.20;
ENDDATA

[Ingresos] Max = @Sum(Pienso(p): precio(p)*sacos(p));
@For(Cereal(c):[Limite] @Sum(Pienso(p): Cantidad(p, c)*sacos(p))<Existencias(c));
```

- La línea de la función objetivo empieza como siempre, con una etiqueta seguida de `Max =`. El resto se lee así: "Suma para todo pienso p del precio de p por el número de sacos producidos de p". En general, dentro de `@Sum()` ponemos el nombre de un conjunto seguido de una nueva variable (en este caso `p`), que hace referencia a un elemento genérico del conjunto, luego `:` y la expresión (en función de `p`) que queremos sumar para todo `p`.

- La segunda línea define simultáneamente cuatro restricciones, una para cada cereal. Podemos leer: "Para todo cereal c, definimos la restricción etiquetada `[Limite]` como que la suma para todo pienso p de la cantidad de cereal c en el pienso p por el número de sacos producidos de p ha de ser $\leq$ que las existencias de c".

  La sintaxis de `@For()` es la misma que la de `@Sum()`. Escribimos el nombre de un conjunto con una nueva variable entre paréntesis (en este caso `c`), luego `:` y luego lo que queremos que LINGO haga para cada valor de `c`.

- Las instrucciones de LINGO que recorren los elementos de un conjunto son las siguientes:

`@For(Conjunto(i): )`	Repite una tarea para todo `i`
`@Sum(Conjunto(i): )`	Suma una expresión para todo `i`
`@Prod(Conjunto(i): )`	Multiplica una expresión para todo `i`
`@Max(Conjunto(i): )`	Calcula el máximo para todo i
`@Min(Conjunto(i): )`	Calcula el mínimo para todo i
`@Writefor(Conjunto(i): )`	Escribe una expresión para todo i (véase la sección 9)

- En general, cuando dentro de estas instrucciones sólo aparece un índice, éste se puede suprimir. Por ejemplo, la función objetivo se podía haber introducido así:

  `[Ingresos] Max = @Sum(Pienso: precio*sacos);`

- Así, si queremos exigir que todas las variables del problema sean enteras podemos escribir

  `@For(Pienso(p): @GIN(Sacos(p)));` o simplemente `@For(Pienso: @GIN(Sacos));`

- Podemos anidar cualquiera de estas instrucciones dentro de otra. Por ejemplo:

  `@Sum(A(i): @Sum(B(j): c(i)*d(j)))`

  calcula la suma de $c_i d_j$ para todo i del conjunto A y todo j del conjunto B.

- Los bucles pueden recorrer conjuntos de pares, ternas etc., en cuyo caso hemos de introducir tantas variables como componentes tiene el conjunto. Por ejemplo, si queremos sumar todas las cantidades para todo pienso y todo cereal, podemos escribir:

  `@Sum(Pareja(p, c): Cantidad(p, c))`

7 Más sobre conjuntos

Es posible definir subconjuntos de un conjunto dado. Por ejemplo:

```
SETS:
Ciudad/C1..C5/:Distancia,Demanda;
Extranjera(ciudad)/C4,C5/:x;
Carretera(Ciudad, Ciudad)|(&1 #LT# &2):y;
ENDSETS

DATA:
Distancia =   0,   50, 1150, 1200, 1500;
Demanda   =1000, 2300,  800, 3000,  900;
ENDDATA

SETS:
Lejana(Ciudad)|(Distancia(&1)#GE#1000):z;
ENDSETS
```

- Aquí hemos definido un conjunto de cinco ciudades `C1, C2, C3, C4, C5`, cada una de las cuales tiene asociada una distancia y una demanda.

- A continuación definimos un subconjunto formado por las ciudades extranjeras, cuyos elementos son `C4, C5`. Para definir un subconjunto escribimos su nombre seguido del nombre del conjunto entre paréntesis.

  Esto significa que, por ejemplo,

  `@Sum(Ciudad(c):Demanda(c))`

  es la suma de las demandas de todas las ciudades, mientras que

  `@Sum(Extranjera(c):Demanda(c))`

  es la suma de las demandas de todas las ciudades extranjeras.

  En general, el hecho de que `Extranjera` sea un subconjunto de `Ciudad` se traduce en que las variables definidas sobre los índices de `Ciudad` están definidas también para los índices de `Extranjera`.

- Un subconjunto se puede definir indicando explícitamente sus elementos (como en el caso de `Extranjera`) o bien mediante una propiedad que los determine. Es lo que hemos hecho para definir el conjunto `Carretera`, que consta de los pares de ciudades (C_i, C_j) con $i < j$. Para entender la definición necesitamos tener presentes varias cosas:

  – LINGO dispone de los operadores lógicos siguientes:

`#EQ#`	igual
`#NE#`	no igual
`#GE#`	mayor o igual
`#GT#`	mayor
`#LT#`	menor
`#LE#`	menor o igual
`#AND#`	y
`#OR#`	o (inclusivo)
`#NOT#`	no

  No hay que confundir el uso de, por ejemplo, `#EQ#` con el de `=`. El primero es un operador lógico que da lugar a expresiones que son verdaderas o falsas, mientras que el segundo da lugar a ecuaciones, como $x = y + 1$, que no son ni verdaderas ni falsas, sino que expresan relaciones entre variables. Siempre que queramos construir una afirmación que deba ser clasificada como verdadera o falsa de forma inmediata, usaremos los operadores anteriores.

  – Para definir un subconjunto mediante una condición lógica se escribe el nombre del subconjunto (en este caso `Carretera`, luego entre paréntesis el nombre del conjunto que lo contiene (en este caso el conjunto de parejas de ciudades), luego el signo `|` y luego la condición lógica (en el ejemplo la hemos puesto entre paréntesis por claridad, pero los paréntesis se pueden suprimir). Luego ya podemos asignar variables al subconjunto, empezando con `:`, como de costumbre (en este caso hemos asignado a cada carretera una variable `y`).

  – Para hacer referencia a un elemento genérico del conjunto en la condición lógica usamos la expresión `&1` (y si es un conjunto de parejas de dos conjuntos A y B usaremos `&1` para un elemento genérico del conjunto A y `&2` para un elemento genérico del conjunto B, etc.)

- Finalmente hemos definido un conjunto `Lejana` formado por las ciudades cuya distancia es mayor o igual que 1 000. Notemos que para definirlo era preciso introducir primero las distancias en una sección `DATA`, por lo que necesariamente hemos tenido que utilizar dos secciones `SETS`.

  Si "resolvemos" el problema obtendremos las variables siguientes:

```
     X( C4)        1.234568
     X( C5)        1.234568
Y( C1, C2)         1.234568
Y( C1, C3)         1.234568
Y( C1, C4)         1.234568
Y( C1, C5)         1.234568
Y( C2, C3)         1.234568
Y( C2, C4)         1.234568
Y( C2, C5)         1.234568
Y( C3, C4)         1.234568
Y( C3, C5)         1.234568
Y( C4, C5)         1.234568
     Z( C3)        1.234568
     Z( C4)        1.234568
     Z( C5)        1.234568
```

Vemos así que las variables `x` están definidas para las ciudades extranjeras, las variables `y` están definidas para los pares de ciudades sin repeticiones y las variables `z` están definidas para las ciudades lejanas.

- También podemos seleccionar elementos de un conjunto a la hora de aplicar cualquiera de los operadores `@For()`, `@Sum`, etc. Por ejemplo

  ```
  @Sum(Ciudad(c)| Demanda(c) #GE# 1000: Distancia(c))
  ```

  suma las distancias a todas las ciudades cuya demanda es mayor o igual que 1 000.

  Notemos que en este contexto no necesitamos los signos `&1, &2,...` porque ya tenemos variables que nombran a los elementos genéricos de los conjuntos (en este caso la variable `c`).

Hay algunas sutilezas sobre la forma en que LINGO entiende los conjuntos que conviene tener presente para no incurrir en confusiones. La clave es que, para LINGO, los elementos de un conjunto son siempre los primeros números naturales. Por ejemplo, si definimos

```
SETS:
Letras/A, C, J, I, L, P/:Puntos,x;
Vocales/A, I/;
ENDSETS

DATA:
Puntos = 1, 3, 4, 6, 9, 12;
ENDDATA
```

En realidad para LINGO el conjunto `Letras` consta de los elementos $\{1, 2, 3, 4, 5\}$, y lo que hemos hecho en la definición es asignar un nombre a cada elemento del conjunto. Así, para LINGO, `A` es el nombre del elemento 1 de `Letras`, y `C` es el nombre del elemento 2, etc.

- Así, por ejemplo, si escribimos la restricción `x(3)=5;` y "resolvemos" el modelo, obtendremos la solución

  ```
  X( A)        1.234568
  X( C)        1.234568
  X( J)        5.000000
  X( I)        1.234568
  X( L)        1.234568
  X( P)        1.234568
  ```

  en la que LINGO ha dado el valor 5 a la variable `X(J)`. En cambio, si escribimos `x(J) = 5;` obtendremos un error, porque, en contra de lo que uno pueda pensar, `J` no es un elemento de `Letras`.

- Si queremos expresar que $x(J) = 5$ sin calcular nosotros mismos el índice correspondiente que hemos de escribir, podemos usar la función `@Index` así:

  ```
  x(@Index(Letras, J)) = 5;
  ```

  En general, `@Index(Conjunto, nombre)` proporciona el elemento del conjunto dado (como número natural) cuyo nombre es el dado.

- Esto es especialmente relevante cuando los nombres de los elementos también son números. Por ejemplo, si definimos `A/2,5,7/:x` y hacemos `x(2) = 10`, LINGO nos dará una solución en la que `x(5)` tomará el valor 10, porque 2 es el elemento de A cuyo nombre es `5` o, dicho de otro modo, `@Index(A, 5)` toma el valor 2.

- Otro ejemplo: si escribimos `z = @Sum(Vocales(v): Puntos(v));`, el valor de `z` que obtendremos no será `Puntos(A)+Puntos(I)` $= 1+6 = 7$, sino 4, porque la variable verb/v/ recorre los elementos de `vocales`, que son los índices 1 y 2, luego la suma es `Puntos(1)+Puntos(2)` $= 1+3 = 4$.

  Esto se debe a que, en realidad, la definición del conjunto `Vocales` no es muy afortunada. Para evitar estas cosas deberíamos haberlo definido como un subconjunto de `Letras`, es decir, así:

  ```
  SETS:
  Letras/A, C, J, I, L, P/:Puntos,x;
  Vocales(Letras)/A, I/;
  ENDSETS
  ```

  De este modo, `Vocales` no es un conjunto con elementos $\{1, 2\}$ (llamados `A, I`), sino que es el subconjunto formado por los elementos de `Letras` llamados `A, I`, luego sus elementos son los números $\{1, 4\}$. Con esta definición, el valor de `z` será el 7 esperado, pues ahora la variable `v` del sumatorio recorrerá los índices 1 y 4.

- Es posible usar `@Index` sin especificar el conjunto, y escribir, por ejemplo `x(@Index(P)) = 10;` y así obtenemos el índice de `P` en el primer conjunto que LINGO encuentra con un elemento llamado `P`. Si no encuentra ninguno se producirá un mensaje de error. Si los conjuntos que manejamos no tienen (nombres de) elementos en común, no hay peligro de suprimir el nombre del conjunto al calcular un índice, pero en caso contrario podemos obtener resultados inesperados.

- Lo fundamental es recordar que las variables que recorren elementos genéricos de conjuntos recorren en realidad números naturales y no los nombres que les hemos asignado en cada conjunto.

- Las funciones básicas para operar con conjuntos son las siguientes:

`@Index(conjunto, nombre)`	proporciona el elemento del conjunto (como número natural) con el nombre dado.
`@In(conjunto, nombre)`	es verdadero si el elemento nombrado está en el conjunto y falso en otro caso.
`@Size(conjunto)`	es el número de elementos del conjunto.

8 Cálculos

A veces el planteamiento de un modelo requiere realizar algunos cálculos con los datos. Tales cálculos se hacen en una sección específica llamada `CALC`. Consideremos por ejemplo el problema siguiente:

Una empresa se plantea la posibilidad de emprender seis proyectos de 5 *años de duración. Se prevé que cada uno de ellos dé lugar a los flujos de caja determinados por la tabla siguiente:*

	Año 1	*Año 2*	*Año 3*	*Año 4*	*Año 5*
Proyecto 1	−1000	−800	900	1000	500
Proyecto 2	−800	−900	−100	1500	1000
Proyecto 3	−1500	100	100	200	1500
Proyecto 4	−1800	−90	−10	1000	2000
Proyecto 5	−2000	900	−1000	1800	1500
Proyecto 6	−950	1000	700	−200	0

La empresa dispone de un presupuesto de 2 000 *u.m. para el primer año y de* 1 800 *u.m. para el segundo año. Para los años siguientes, las inversiones requeridas por cada proyecto deberán financiarse con los beneficios proporcionados por los otros. Por otra parte, la empresa puede retrasar el inicio de cada proyecto hasta un máximo de* 2 *años. Determinar los proyectos que debe emprender la empresa y el año en que debe iniciarlos para maximizar el VAN total de su inversión, considerando un factor de capitalización* $k = 0.04$.

Introducimos los datos en el modelo del modo siguiente:

```
SETS:
Proyecto/1..6/;
Periodo/1..5/;
Pareja(Proyecto,Periodo):Flujo;
Anyo/1..8/:presupuesto;
ENDSETS

DATA:
k = 0.04;
Flujo =      -1000  -800   900  1000   500
              -800  -900  -100  1500  1000
             -1500   100   100   200  1500
             -1800   -90   -10  1000  2000
             -2000   900 -1000  1800  1500
              -950  1000   500  -200   900;
Presupuesto = 2000, 1800,    0,    0,    0, 0, 0;
ENDDATA
```

Notemos que hemos definido un conjunto de 7 años porque, como cada proyecto puede retrasarse hasta un máximo de 2 años, el proyecto global puede durar hasta un máximo de 7 años. Vamos a desdoblar cada proyecto en tres proyectos distintos, según que se empiece a efectuar en el primer año, en el segundo o en el tercero. Para ello necesitamos más conjuntos:

```
SETS:
Proyecto/1..6/;
Periodo/1..5/;
Pareja(Proyecto,Periodo):Flujo;
Anyo/1..7/:presupuesto;
Inicio/1..3/;
Terna(Proyecto, inicio,Anyo):Fl;
ProyIni(Proyecto, inicio):Van, x
```

Las variables `Fl(p, i, a)` contendrán el flujo de caja que genera en el año `a` el proyecto `p` si se inicia en el año `i`. Las variables `Van(p, i)` contendrán el VAN del proyecto `p` si se inicia en el año `i`, mientras que las variables `x(p,i)` serán las auténticas variables del problema, variables binarias que determinarán si se inicia o no el proyecto `p` en el año `i`. Introducimos entonces una sección `CALC` para calcular los valores `Fl(p, i, a)` y `Van(p,i)`:

```
CALC:
@For(Terna(p,i,a):
     Fl(p,i,a) = @IF(a-i+1 #GE# 1 #AND# a-i+1 #LE# @Size(Periodo), Flujo(p,a-i+1),0));
@For(ProyIni(p,i): Van(p,i) = @Sum(Anyo(a): (1+k)^(1-a)*Fl(p,i,a)));
ENDCALC
```

Para calcular los flujos hemos usado la función `@IF(condicion, casoV, casoF)`, que proporciona como resultado la expresión `casoV` si la `condicion` es verdadera y la expresión `casoF` si es falsa. Hemos usado que el flujo en el año `a` del proyecto `p` iniciado en el año `i` es el flujo del proyecto `p` en el periodo `a-i+1`, salvo que este índice no esté entre 1 y 5, en cuyo caso el flujo es 0.

Por definición, el VAN es la suma de los flujos de caja transportados al año inicial multiplicándolos por el factor $(1+k)^{-t}$.

- Si hubiéramos intentado incluir estas líneas en la sección `DATA` habríamos obtenido un error, pues LINGO no permite hacer cálculos con bucles etc. en una sección `DATA`.

- Otra diferencia es que en una sección `DATA` cada variable puede ser definida sólo una vez, mientras que en una sección `CALC` podemos modificar el valor de una variable definida en una sección `DATA`, o definir varias veces la misma variable, o dar definiciones de una variable en términos de ella misma, como

  ```
  x = 2*y+z;
  x = x/100;
  ```

  Notemos que si escribiéramos estas líneas fuera de cualquier sección, LINGO las tomaría por restricciones y de la segunda concluiría que $x = 0$.

- Más aún, en una seccion `CALC` podemos hacer que una variable a la que se le ha asignado un valor fijo (sea en una sección `DATA` o `CALC`) vuelva a ser considerada como variable, mediante la instrucción `@Release(x);`

Ahora ya es fácil escribir las restricciones. El modelo completo queda así:

```
SETS:
Proyecto/1..6/;
Periodo/1..5/;
Pareja(Proyecto,Periodo):Flujo;
Anyo/1..7/:presupuesto;
Inicio/1..3/;
Terna(Proyecto, inicio,Anyo):Fl;
ProyIni(Proyecto, inicio):Van, x;
ENDSETS
DATA:
k = 0.04;
Flujo =      -1000  -800   900  1000   500
              -800  -900  -100  1500  1000
             -1500   100   100   200  1500
             -1800   -90   -10  1000  2000
             -2000   900 -1000  1800  1500
              -950  1000   500  -200   900;
Presupuesto = 2000, 1800,    0,    0,    0, 0, 0;
ENDDATA

CALC:
@For(Terna(p, i,a):
     Fl(p,i,a) = @If(a-i+1#GE# 1 #AND# a-i+1 #LE# Size(Periodo),Flujo(p,a-i+1),0));
@For(ProyIni(p,i): Van(p,i) = @Sum(Anyo(a): (1+k)^(1-a)*Fl(p,i,a)));
ENDCALC

[VANTotal] Max = @Sum(ProyIni(p, i): Van(p,i)*x(p, i));
@For(Anyo(a):[ResPres] @Sum(ProyIni(p, i): Fl(p,i,a)*x(p,i))+presupuesto(a)>0);
@For(proyecto(p):[UnIni]@Sum(inicio(i): x(p, i))<1);
@For(ProyIni: @BIN(x));
```

Notemos que las restricciones `UniIni` exigen que cada proyecto se inicie solamente una vez.

En una sección `CALC` podemos usar la instrucción `@SET` para establecer las opciones que pueden establecerse manualmente desde el menú LINGO → Options... De este modo nos aseguramos de que nuestro modelo no se comporte de forma diferente en distintos ordenadores por tener configuraciones distintas por defecto. La sintaxis general es `@SET('nombre',valor);`, excepto en el caso `@SET('Default');` que restaura las opciones por defecto y no requiere ningún parámetro. Destacamos algunas de las opciones:

Nombre	Por defecto	Descripción
Default	–	Restablece las opciones por defecto.
Dualco	1	Cálculos duales: 0 (**none**) Ninguno 1 (**prices only**) Precios duales 2 (**prices & ranges** Precios duales e intervalos de sensibilidad)
Global	0	1 busca óptimos globales, 0 no.
Nonneg	1	1 supone las variables no negativas, 0 no.
Unarym	0	$0 \to -2^2 = 4, \quad 1 \to -2^2 = -4$
Linear	0	1 supone que el modelo es lineal, 0 no.
Terseo	0	Determina qué escribe LINGO en su informe .lrg: 0 (**verbose**) es la opción por defecto 1 (**terse**) Imprime el valor de la función objetivo y poco más 2 (**errors only)** Sólo imprime los errores 3 (**nothing**) Nada
Errdlg	1	1 muestra cuadros de diálogo con errores, 0 no.

9 Personalización de la salida

Cuando modelizamos un problema con conjuntos, LINGO escribe en su informe .lgr los valores de todas las variables, contando entre ellas los datos del problema, con lo que los valores más relevantes pueden quedar perdidos en un mar de números. Para evitar esto podemos personalizar el formato del informe. También podemos hacer que LINGO escriba parte de la información del problema en un archivo que pueda a su vez ser leído por otra aplicación para procesar los resultados.

Sin embargo, a la hora de hacer esto nos encontramos con un inconveniente, y es que LINGO ejecuta primero todas las instrucciones que encuentra en el modelo y finalmente resuelve el problema, con lo que, en principio, no nos da opción a pedirle que escriba nada relacionado con la solución (si le pedimos que escriba el valor óptimo de las variables, lo hará antes de haber resuelto el problema, y no nos dirá nada de interés). Para evitar esto debemos definir el problema como un submodelo.

Consideremos por ejemplo el problema estudiado en la sección anterior, que ahora retocamos así:

```
SETS:
Proyecto/1..6/:ini;
Periodo/1..5/;
Pareja(Proyecto,Periodo):Flujo;
Anyo/1..7/:presupuesto;
Inicio/1..3/;
Terna(Proyecto, inicio,Anyo):Fl;
ProyIni(Proyecto, inicio):Van, x;
ENDSETS
DATA:
k = 0.04;
Flujo = -1000  -800   900  1000   500
         -800  -900  -100  1500  1000
        -1500   100   100   200  1500
        -1800   -90   -10  1000  2000
        -2000   900 -1000  1800  1500
         -950  1000   700  -200     0;
Presupuesto = 2000,1800,0,0,0,0,0;
ini = 0;
ENDDATA
```

```
SUBMODEL Proyectos:
[VANTotal] Max = @Sum(ProyIni(p, i): Van(p,i)*x(p, i));
@For(Anyo(a):[ResPres] @Sum(ProyIni(p, i): Fl(p,i,a)*x(p,i))+presupuesto(a)>0);
@For(proyecto(p):[UnIni]@Sum(inicio(i): x(p, i))<1);
@For(ProyIni: @BIN(x));
ENDSUBMODEL

CALC:
@For(Terna(p, i,a):
     Fl(p,i,a) = @If(a-i+1#GE# 1 #AND# a-i+1 #LE# @Size(Periodo),Flujo(p,a-i+1),0));
@For(ProyIni(p,i): Van(p,i) = @Sum(Anyo(a): (1+k)^(1-a)*Fl(p,i,a)));

@SOLVE(Proyectos);
ENDCALC
```

Hemos añadido unas variables `ini(p)` para cada proyecto a las que hemos dado el valor 0. Salvo por esto, el resultado es el mismo que antes, sólo que ahora las ecuaciones del modelo están incluidas en una sección `SUBMODEL`. A diferencia de las demás secciones, una sección `SUBMODEL` ha de tener necesariamente un título, que en este caso es `Proyectos`. Como vemos, el título se pone entre la palabra `SUBMODEL` y los dos puntos. El efecto de esto es que LINGO no resuelve el modelo mientras no se encuentre la instrucción `@SOLVE(Proyectos)`. Si eliminámaramos la penúltima línea, LINGO no haría nada.

De este modo tenemos localizado el momento en que LINGO resuelve el problema. Por ejemplo, si escribimos

```
@Write(x(1,1));
@SOLVE(Proyectos);
@Write(x(1,1));
```

Nos encontraremos con que LINGO escribe el valor `1.234567880630493` antes de la solución óptima (es decir, el valor de la variable `x(1,1)` antes de que LINGO resuelva el problema) y el valor `1` después de la solución óptima, pues éste es, en efecto, el valor óptimo de dicha variable. Para personalizar el informe con la solución escribimos lo siguiente:

```
@SET('terseo',3);
@SOLVE(Proyectos);
@For(Proyecto(p): ini(p) = @Sum(inicio(i)|x(p,i) #EQ# 1:i));

@Write('Año',14*' ');
@WriteFor(Anyo(a):a,9*' ');
@Write(@Newline(1),(15+@Size(Anyo)*10)*'-',@Newline(1));
@For(Proyecto(p):      !Aquí empieza un bucle.
@Write('Proyecto ',p);
@IFC(ini(p)#NE#0:      !Aquí empieza un condicional;
@Write(10*(ini(p)-1)*' ');
@WriteFor(Periodo(pr): @Format(Flujo(p,pr),'10.0f'));
);                     !Aquí termina el condicional;
@Write(@Newline(1)));  !Aquí termina el bucle;
@Write((15+@Size(Anyo)*10)*'-',@Newline(1),'Flujo:    ');
@WriteFor(Anyo(a): @Format(Respres(a)-Presupuesto(a),'10.0f'));
@Write(@Newline(2),'VAN: ', @Format(VANTotal,'0.2f'));
```

Antes de explicar el significado de estas instrucciones, veamos el resultado:

```
Año              1         2         3         4         5         6         7
-------------------------------------------------------------------------------
Proyecto 1     -1000      -800       900      1000       500
Proyecto 2
Proyecto 3                       -1500       100       100       200      1500
Proyecto 4               -1800      -90       -10      1000      2000
Proyecto 5
Proyecto 6      -950      1000       700      -200         0
-------------------------------------------------------------------------------
Flujo:         -1950     -1600        10       890      1600      2200      1500

VAN: 1673.39
```

- La primera instrucción hace que LINGO no escriba nada por su cuenta en el informe; la siguiente resuelve el problema, y la siguiente calcula el valor de `ini(p)` como el año de inicio del proyecto `p` (que conserva su valor 0 si el proyecto no se ejecuta).

- La instrucción básica para escribir es `@Write()`, dentro de la cual ponemos una sucesión de números y/o cadenas de signos separados por comas. Toda cadena de signos se escribe entre comillas simples o dobles.

- Así, la instrucción `@Write('Año',14*' ');` escribe la palabra "Año" seguida de 14 espacios en blanco (LINGO admite la multiplicación de un número natural n por una cadena, cuyo resultado es la cadena repetida n veces).

- La instrucción `@WriteFor(Anyo(a):a,9*' ');` genera un bucle de escritura. Para cada año `a`, escribe el valor de `a` seguido de nueve espacios en blanco.

- La instrucción `@Write(@Newline(1),(15+@Size(Anyo)*10)*'-',@Newline(1));` escribe un cambio de línea, una cantidad de signos `-` proporcional al número de años y otro cambio de línea. En general, la instrucción `@Newline(n)` inserta n cambios de línea. Con esto tenemos escritas las dos primeras líneas del informe.

- A continuación empieza un bucle `@For(Proyecto(p): );` que termina seis líneas más abajo. Para cada valor de `p` se imprime una línea del informe.

- La línea correspondiente al proyecto `p` empieza con el texto "Proyecto p", que se escribe con la instrucción `@Write('Proyecto ',p);`

- El resto de la línea contendrá datos o estará en blanco en función de si el proyecto `p` se ejecuta o no. Para distinguir ambos casos usamos la instrucción `@IFC(ini(p)#NE#0: );` que termina tres líneas más abajo.

  No hemos de confundir la instrucción `@IF(condicion, casoV, casoF)`, que proporciona un resultado, un cálculo u otro según que la condición sea verdadera o falsa, con la instrucción

  `@IFC(condicion, instrucciones)`;

  que realiza las instrucciones dadas si y sólo si la condición es verdadera. La estudiaremos con más detalle en la sección 12. En nuestro caso, resulta que LINGO ejecuta las dos instrucciones siguientes si y sólo si el proyecto `p` se ejecuta.

- La instrucción `@Write(10*(ini(p)-1)*' ');` escribe una cantidad de espacios en blanco proporcional al año de inicio del proyecto.

- La instrucción `@WriteFor(Periodo(pr): @Format(Flujo(p,pr),'10.0f'));` ejecuta un bucle de escritura. Para cada uno de los cinco periodos `pr` escribe el flujo del proyecto `p` correspondiente al periodo `pr`. La instrucción `@Format()` especifica la forma en que se escribe un número o una cadena. Aquí estamos especificando que escriba el flujo en un espacio de 10 caracteres con 0 decimales. Luego describiremos todos los formatos posibles.

- Tanto si el proyecto se ejecuta como si no, terminamos la línea con un cambio de línea dado por la instrucción `@Write(@Newline(1))`. Con ella termina el paso del bucle correspondiente al proyecto `p`.

- A continuación escribimos una línea de signos `-` de longitud proporcional al número de años, luego un cambio de línea, y luego la palabra "Flujo:".

- A continuación un nuevo bucle escribe, para cada año `a`, la variable de holgura de la restricción presupuestaria de dicho año menos el presupuesto de dicho año, lo cual es el flujo de caja total de dicho año.

- Finalmente escribimos dos cambios de línea, la palabra "VAN: " y el valor óptimo de la función objetivo (con el formato adecuado).

Nos falta explicar con detalle el uso de `@Format()` y el modo de acceder a toda la información relacionada con la solución óptima (sea para escribirla, sea para manipularla).

En general, para formatear un número escribimos `@Format(numero, 'formato')`, donde `formato` (escrito entre comillas) es una cadena de signos dentro de las posibilidades indicadas en la tabla siguiente:

Formato	Descripción
10.5f	Ocupa 10 espacios con 5 decimales
10.5e	Notación científica en 10 espacios con 5 decimales
10.5g	Notación decimal o científica en 10 espacios con 5 cifras significativas
#10.5g	Igual que el anterior, pero si hay menos cifras significativas las completa con ceros.

También se puede usar `@Format()` con una cadena de signos, en cuyo caso los formatos posibles son dos: 10s la escribe en 10 espacios justificada a la izquierda, y -10s la escribe en 10 espacios justificada a la derecha.

Para hacer cálculos sobre la disposición del texto puede ser útil la función `@Strlen(cadena)`, que da la longitud de la cadena.

Cuando el número o la cadena no cabe en el espacio señalado, se sale por la derecha. Así, por ejemplo, la instrucción `@Format(VANTotal,'0.2f')` no deja ningún espacio para escribir la función objetivo, por lo que ésta se escribe inmediatamente a continuación, sin dejar ningún espacio en blanco (con dos decimales).

- Para referirnos a cualquier variable (en particular, para escribirla) usamos su nombre. Así, por ejemplo, para escribir el VAN del primer proyecto si empieza en el año 1 escribiremos `@Write(Van(1,1));`

- Para referirnos a una variable de holgura o a la función objetivo usaremos la etiqueta de su restricción. Por ejemplo, ya hemos visto que `@Write(VANTotal);` imprime el valor óptimo de la función objetivo, y `@Write(Respres(1));` escribe la variable de holgura de la restricción presupuestaria del año 1.

- Además podemos aplicar las funciones siguientes al nombre de cualquier variable:

Nombre	Resultado
`@Name()`	Nombre de la variable
`@Dual()`	Variable dual (coste reducido o precio dual)
`@RangeD()`	Disminución máxima en el intervalo de sensibilidad
`@RangeU()`	Aumento máximo en el intervalo de sensibilidad

- Finalmente, la función `@Status()` (sin nada dentro de los paréntesis) nos proporciona un código que corresponde al resultado del proceso de resolución, de acuerdo con la tabla siguiente:

Código	Significado
0	Óptimo global
1	Infactible
2	No acotado
3	Indeterminado (LINGO no ha sabido resolver el problema)
4	Factible
5	Infactible o no acotado
6	Óptimo local
7	Localmente infactible
8	Se ha alcanzado el nivel de precisión exigido
9	Error numérico

Si escribimos las líneas

```
@Divert('archivo.txt');
...
@Divert();
```

LINGO creará un archivo llamado `archivo.txt` e imprimirá en él todo lo que tenga que imprimir (incluido su informe habitual, si no lo hemos cancelado), hasta que se encuentre con la instrucción `@Divert();`

Podemos anidar varias instrucciones de este tipo con distintos nombres de documentos. Cada vez que LINGO encuentra una instrucción `@Divert();` cierra el archivo en el que está escribiendo y pasa a escribir en el que escribía antes de haber abierto el último, hasta que ya no queda ninguno abierto, y entonces pasa a escribir en la pantalla (en un documento .lgr).

10 Submodelos

LINGO permite definir varios submodelos parciales, de modo que la instrucción `@Solve` puede tomar como argumentos varios submodelos a la vez, con la única condición de que entre todos no haya más de una función objetivo. Veamos un ejemplo de la utilidad de esta posibilidad.

El modelo siguiente define un conjunto de 12 escenarios y, para cada uno de ellos, tiene como datos las rentabilidades esperadas de tres activos financieros llamados `ATT, GMT, USX`. Cada escenario tiene asociada una probabilidad, que en el ejemplo es la misma para todos ellos, igual a 1/12. El problema consiste en elegir la proporción `X(i)` en que cada uno de ellos debe aparecer en una cartera de inversión para garantizar una rentabilidad esperada de al menos 1.15% con el mínimo riesgo. Se proponen tres medidas de riesgo diferentes:

1. La varianza, que es la media ponderada (con las probabilidades) de los cuadrados de las desviaciones de la rentabilidad esperada para la cartera en cada escenario respecto de la rentabilidad media en todos los escenarios.

2. La semivarianza, en la que sólo se tienen en cuenta las desviaciones negativas, es decir, las de los escenarios en los que la rentabilidad esperada queda por debajo de la media.

3. El riesgo inferior, en el que la media ponderada se hace también considerando las desviaciones negativas, pero sin elevarlas al cuadrado.

Como vemos, se define un submodelo con las restricciones (en el que, por conveniencia, se calculan las tres medidas de riesgo) y luego otros tres submodelos en los que se escoge una de las funciones como función objetivo. Cada instrucción `@Solve` incluye el submodelo de las restricciones y uno de los objetivos.

```
SETS:
   Escenario/1..12/: Prob, R, DesvSup, DesvInf;
   Activo/ ATT,  GMT,  USX/:  X;
   Par(Escenario, Activo): Rent;
ENDSETS
DATA:
   Objetivo = 1.15;
   Rent =
    1.300    1.225    1.149
    1.103    1.290    1.260
    1.216    1.216    1.419
    0.954    0.728    0.922
    0.929    1.144    1.169
    1.056    1.107    0.965
    1.038    1.321    1.133
    1.089    1.305    1.732
    1.090    1.195    1.021
    1.083    1.390    1.131
    1.035    0.928    1.006
    1.176    1.715    1.908;
   Prob= .08333;
ENDDATA

Submodel Restricciones:
!Esta ecuación hace que Media sea la media ponderada
sobre todos los escenarios e de las rentabilidades R(e);
Media = @SUM(Escenario: Prob * R);
Media >= Objetivo;
@SUM(activo: X) =1;

@FOR(Escenario(e):
!Aquí definimos la rentabilidad de la cartera en el escenario e
como la media de las rentabilidades de los activos ponderada por
el peso de cada uno en la cartera;

R(e) = @SUM(activo(i): Rent(e, i) * X(i));

!La ecuación siguiente define las desviaciones superior e inferior
respecto a la media salvo una constante, pero el objetivo de minimizar
hará que la constante se ajuste para que al menos una de las dos sea nula;

DesvSup(e) - DesvInf(e) = R(e) - Media);

[Varianza]       Var       = @SUM(Escenario: Prob * (DesvSup + DesvInf)^2);
[Semivarianza]   Semivar   = @SUM(Escenario: Prob * DesvInf^2);
[RiesgoInferior] RiesgoInf = @SUM(Escenario: Prob * DesvInf);
EndSubmodel

Submodel MinVar:
[MV] Min = Var;
EndSubmodel
```

```
Submodel MinSemi:
[MS] Min = Semivar;
EndSubmodel

Submodel MinRI:
[MR] Min = RiesgoInf;
EndSubmodel

CALC:
@SET('Terseo',3);
@Solve(Restricciones, MinVar);
@Write(11*' ');
@WriteFor(Activo:@Format(@Name(x),'8s'));
@Write('   Var   Semivar  RiesgoInf',@Newline(1));
@Write('MinVar:   ');
@WriteFor(Activo: @Format(X,'8.3f'));
@Write(@Format(Var,'8.3f'),@Format(SemiVar,'8.3f'), @Format(RiesgoInf,'8.3f'),@Newline(1));
@Solve(Restricciones, MinSemi);
@Write('MinSemi:  ');
@WriteFor(Activo: @Format(X,'8.3f'));
@Write(@Format(Var,'8.3f'),@Format(SemiVar,'8.3f'), @Format(RiesgoInf,'8.3f'),@Newline(1));
@Solve(Restricciones, MinRi);
@Write('MinRI:    ');
@WriteFor(Activo: @Format(X,'8.3f'));
@Write(@Format(Var,'8.3f'),@Format(SemiVar,'8.3f'), @Format(RiesgoInf,'8.3f'),@Newline(1));
ENDCALC
```

El resultado es el siguiente:

```
            X( ATT) X( GMT) X( USX)   Var   Semivar  RiesgoInf
MinVar:      0.530   0.357   0.114   0.021   0.010   0.056
MinSemi:     0.575   0.039   0.386   0.024   0.009   0.065
MinRI:       0.511   0.489   0.000   0.021   0.012   0.056
```

Vemos que las distintas medidas del riesgo llevan a soluciones óptimas diferentes.

11 Lectura de datos en ficheros

La instrucción para leer datos de un fichero es `@FILE('nombre del fichero')`. La primera vez que aparece esta instrucción LINGO abre el fichero y lee los datos hasta que aparezca el carácter `~`, y cada sucesiva aparición de la instrucción hace que LINGO lea los datos que siguen hasta la próxima aparición del carácter `~`. Si en lugar de dicho carácter se acaba el fichero, LINGO cierra el archivo. Los datos en el fichero han de estar exactamente igual como estarían si los escribiéramos directamente en el modelo. Por ejemplo, en lugar de

```
SETS:
Ciudad/Barcelona, Madrid, Valencia/:demanda;
Fabrica/1..20/:oferta;
ENDSETS
```

podemos escribir:

```
SETS:
Ciudad/@FILE('datos.txt')/:demanda;
Fabrica/@FILE('datos.txt')/:oferta;
ENDSETS
```

e importar los datos de un archivo llamado `datos.txt` que contenga el texto siguiente:

```
Barcelona, Madrid, Valencia ~
1..20
```

o también vale

```
SETS:
Ciudad/@FILE('datos.txt')/:demanda;
Fabrica/1..@FILE('datos.txt')/:oferta;
ENDSETS
```

si en el archivo de datos escribiemos `20` en lugar de `1..20`, o incluso

```
SETS:
Ciudad/@FILE('datos.txt')/:demanda;
@FILE('datos.txt')
ENDSETS
```

si en el archivo de datos escribimos:

```
Barcelona, Madrid, Valencia ~
Fabrica/1..20/:oferta;
```

(Notemos que la última línea no termina en `;` porque el `;` está ya en el archivo y LINGO lo lee igualmente.) En suma, cualquier porción de un modelo puede ser leída de un archivo de texto.

NOTA: Los archivos han de tener periódicamente cambios de línea. LINGO no los tendrá en cuenta para nada, pero no leerá líneas demasiado largas.

12 Control del flujo

A la hora de dirigir la actividad de LINGO disponemos de varias instrucciones.

- Ya conocemos `@FOR()`, que nos permite incluir una lista de instrucciones separadas por `;` que se ejecutarán para cada elemento de un conjunto prefijado.
- Una instrucción similar es `@WHILE()`, en la que el bucle no depende de un conjunto sino de una condición lógica. La sintaxis es:

  ```
  @WILE(condicion: instrucciones);
  ```

  Las instrucciones se repetirán hasta que deje de cumplirse la condición.
- Si en el interior de un bucle `@For` o `@While` LINGO se encuentra la instrucción `@BREAK` saldrá del bucle inmediatamente.
- La instrucción `@STOP('mensaje de error')` abre un cuadro de diálogo con el mensaje de error y detiene el cálculo.
- La instrucción `@PAUSE('texto')` muestra un mensajde y da la opción de continuar o detener el cálculo. El mensaje puede combinar cadenas de texto y números como una instrucción `@WRITE`.

- También hemos visto antes la instrucción `@IFC`. Se puede usar opcionalmente en combinación con `@ELSE`. Un ejemplo sencillo sería el siguiente:

```
DATA:
r=?;
ENDDATA

Submodel Nada:
x=0;
Endsubmodel

CALC:
@Solve(Nada);
@IFC(r #GE#100:
@PAUSE('El número es grande');
@ELSE
@PAUSE('El número es pequeño')
);
ENDCALC
```

  En primer lugar notemos que si en una sección `DATA` un dato aparece igualado a `?`, LINGO pregunta por él mediante un cuadro de diálogo en el momento en que tiene que resolver un modelo (pero no antes, por eso hemos tenido que crear un modelo tonto, para forzar a LINGO a preguntar por `r`).

  Luego tenemos una instrucción `@IFC` cuya condición es $r \geq 100$. Si se cumple LINGO ejecuta una única instrucción (podriamos haber puesto varias), que muestra un cuadro de diálogo. A continuación escribimos `@ELSE`, cuyo efecto es que las instrucciones que vienen después se ejecutan sólo si la condición es falsa. En este caso LINGO muestra otro cuadro de diálogo distinto.