CAPÍTULO 11

INVESTIGACIÓN OPERATIVA CON *R*

1.- PROGRAMACIÓN LINEAL

2.- PROGRAMACIÓN LINEAL ENTERA

3.- LOS PROBLEMAS DE TRANSPORTE Y DE ASIGNACIÓN

4.- OTRAS OPCIONES PARA RESOLVER PROBLEMAS DE PROGRAMACIÓN LINEAL EN *R*

1.- PROGRAMACIÓN LINEAL

R es, en esencia, un software para hacer estadística. No obstante, y teniendo en cuenta que se trata de un software de código libre, multitud de aportaciones de diversos ámbitos se hacen continuamente. Entre ellas, y dado que en algunos aspectos existe un importante contacto entre la Estadística y la Investigación Operativa también es posible utilizar *R* para resolver cuestiones de esta ultima materia.

La Investigación Operativa puede considerarse como una especie de miscelánea de diversas materias que, en ocasiones, no tienen mucho en común. El núcleo básico lo constituye la Programación Lineal (PL) y es a este único aspecto al que vamos a dedicar la atención en este capítulo, aunque muy someramente.

En los problemas PL se trata de optimizar una función objetivo lineal de *n* variables de la forma

$$c_1x_1 + c_2x_2 + ... + c_nx_n$$

sujeta a un conjunto de restricciones también lineales:

$$\sum_{j=1}^{m} a1_{ij} x_j \leq b1_j$$

$$\sum_{j=1}^{r} a2_{ij} x_j \leq b2_j$$

$$\sum_{j=1}^{s} a3_{ij} x_j = b3_j$$

$$x_i \geq 0, \forall i$$

Para empezar vamos a cargar el paquete **`boot`**. En realidad se trata de un paquete que contiene funciones y datos para técnicas de estadística robusta pero que contiene así mismo la función **`simplex`** para resolver problemas PL. Los argumentos de esta función son, en este orden:

`a:` vector formado por los coeficientes de la función objetivo,
`A1:` matriz de coeficientes del lado izquierdo de las restricciones del tipo ≤,
`b1:` vector de coeficientes del lado derecho de las restricciones del tipo ≤,
`A2:` matriz de coeficientes del lado izquierdo de las restricciones del tipo ≥,
`B2:` vector de coeficientes del lado derecho de las restricciones del tipo ≥,
`A3:` matriz de coeficientes del lado izquierdo de las restricciones del tipo =,
`B3:` vector de coeficientes del lado derecho de las restricciones del tipo =,
`maxi:` FALSE (por defecto) si se trata de un problema de minimización y TRUE si se trata de un problema de maximización.

Veamos a continuación dos ejemplos.

Ejemplo 1: Resolver el problema de programación lineal:

$$\begin{aligned}
\text{Max} \quad Z &= 60x_1 + 35x_2 + 20x_3 \\
8x_1 + 6x_2 + x_3 &\leq 48 \\
4x_1 + 2x_2 + 1.5x_3 &\leq 20 \\
2x_1 + 1.5x_2 + 0.5x_3 &\leq 8 \\
x_2 &\leq 5 \\
x_1, x_2, x_3 &\geq 0
\end{aligned}$$

```
> library(boot)
> a<-c(60,35,20)
> coef.men.ig<-c(8,6,1,4,2,1.5,2,1.5,0.5,0,1,0)
> A1<-matrix(coef.men.ig,nrow=4,byrow=T)
> A1
     [,1] [,2] [,3]
[1,]    8  6.0  1.0
[2,]    4  2.0  1.5
[3,]    2  1.5  0.5
[4,]    0  1.0  0.0
> b1<-c(48,20,8,5)
> simplex(a,A1,b1,maxi=T)

Linear Programming Results

Call : simplex(a = a, A1 = A1, b1 = b1, maxi = T)

Maximization Problem with Objective Function
Coefficients
x1 x2 x3
60 35 20


Optimal solution has the following values
x1 x2 x3
 2  0  8
The optimal value of the objective  function is
280.
```

La salida se interpreta del siguiente modo: los valores óptimos de las variables son, respectivamente, 2, 0 y 8; el óptimo de la función objetivo es 280. El problema admite, en este caso, infinitas soluciones, como por ejemplo: x_1=0, x_2=1.6, x_3=11.2. No hay información sobre este aspecto en la salida de la función.

Ejemplo 2: Resolver el problema de programación lineal:

$$
\begin{array}{ll}
\text{M}in & Z = 800x_1 + 400x_2 + 600x_3 + 500x_4 \\
& 10x_1 + 3x_2 + 8x_3 + 2x_4 \geq 5 \\
& 90x_1 + 150x_2 + 75x_3 + 175x_4 \geq 100 \\
& 45x_1 + 25x_2 + 20x_3 + 37x_4 \geq 30 \\
& x_1 + x_2 + x_3 + x_4 = 1 \\
& x_1, x_2, x_3, x_4 \geq 0
\end{array}
$$

```
> library(boot)
> a<-c(800,400,600,500)
>                              coef.mayor.o.igual<-
c(10,3,8,2,90,150,75,175,45,25,20,37)
> A1<-c(rep(0,4))
> A1
[1] 0 0 0 0
> b1<-c(0)
> b1
[1] 0
> A2<-matrix(coef.mayor.o.igual,nrow=3,byrow=T)
> A2
     [,1] [,2] [,3] [,4]
[1,]   10    3    8    2
[2,]   90  150   75  175
[3,]   45   25   20   37
> b2<-c(5,100,30)
> coef.igual<-c(1,1,1,1)
> A3<-matrix(coef.igual,nrow=1,byrow=T)
> A3
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
>
> b3<-c(1)
> simplex(a,A1,b1,A2,b2,A3,b3)

Linear Programming Results

Call : simplex(a = a, A1 = A1, b1 = b1, A2 = A2,
b2 = b2, A3 = A3, b3 = b3)

Minimization   Problem   with   Objective   Function
Coefficients
 x1  x2  x3  x4
800 400 600 500


Optimal solution has the following values
        x1         x2         x3         x4
0.25925926 0.70370370 0.03703704 0.00000000
```

```
The optimal value of the objective  function is
511.111111111111.
```

Se ha de hacer notar aquí que, al no haber restricciones del tipo menor o igual, hemos debido generar una restricción ficticia formada por una matriz de ceros **`A1`** y un vector nulo **`b1`**.

El método de cómputo que emplea *R* al utilizar la función **`simplex`** es adecuado sólo para problemas de relativamente pequeño tamaño. Si es posible, el número de restricciones debería ser reducido al mínimo posible, pues el tiempo de ejecución es aproximadamente proporcional al cubo del número de restricciones. En particular, si hubiera restricciones del tipo x_i>=$b2_i$ deberían ser omitidas haciendo x_i=x_i-$b2_i$, cambiando todas las restricciones y la función objetivo como corresponda y haciendo la transformación inversa una vez obtenida la solución.

2.- PROGRAMACIÓN LINEAL ENTERA

En algunos problemas de programación lineal las variables toman valores en un conjunto finito o infinito numerable (un caso particular es cuando las variables son binarias tomando sólo los valores cero o uno). En este caso el problema se convierte en un problema de Programación Lineal Entera (PLE). Aunque a simple vista el problema parece ser más simple, lo cierto es que las dificultades se incrementan, existiendo diferentes algoritmos para su resolución.

Para resolver en *R* este tipo de problemas se puede usar el paquete **`lpSolve`**. Se trata de un software para resolver problemas de programación lineal, entera y mixta (cuando algunas variables deben ser enteras y otras no). También se pueden resolver algunos problemas especiales de programación lineal que abordaremos en el siguiente epígrafe. En concreto, para resolver problemas PLE debemos utilizar la función **`lp`**. Veamos un ejemplo a continuación.

Ejemplo 3: Resolver el problema de programación lineal entera:

$$\begin{aligned} \text{Max} \quad & Z = 5x_1 + 7x_2 \\ & 8x_1 + 14x_2 \leq 63 \\ & 10x_1 + 4x_2 \leq 45 \\ & x_1, x_2 \geq 0 \text{ y enteras} \end{aligned}$$

```
> library(lpSolve)
> coef.obj<-c(5,7)
>   coef.res.li<-matrix   (c(8,14,10,4),   nrow=2,
byrow=TRUE)
>  #En  la  matriz  anterior  se  incluyen  los
coeficientes  del  lado  izquierdo  de  todas  las
restricciones, sean del tipo que sean
```

```
> res.dir <- c("<=","<=")#Vector de direcciones de
todas las desigualdades
> coef.res.ld<-c(63,45)#Coeficientes del lado
derecho de todas las restricciones
>
lp(direction="max",objective.in=coef.obj,const.mat
=coef.res.li,
+const.dir=res.dir,const.rhs=coef.res.ld,int.vec=c
(1,2),all.int=T)
Success: the objective function is 31
>#El argumento int.vec le indica a la función, a
través de un vector, las variables que deben ser
enteras (x1 y x2) y con all.int=T se señala que
todas las variables deben ser enteras, que no es
un problema de programación lineal entera mixta

>#Para conocer para qué valores de las variables
se obtiene el óptimo 31, hacemos lo siguiente
>
lp(direction="max",objective.in=coef.obj,const.mat
=coef.res.li,
+
const.dir=res.dir,const.rhs=coef.res.ld,int.vec=c(
1,2),all.int=T)$solution
[1] 2 3
```

La solución al problema es x_1=2, x_2=3 y $Z_{óptimo}$=31.

3.- LOS PROBLEMAS DE TRANSPORTE Y DE ASIGNACIÓN

Vamos a abordar dos problemas especiales de programación lineal para los que existen algoritmos específicos de resolución y que suelen aparecer en muchas aplicaciones prácticas. Se trata del "*problema del transporte*" y del "*problema de asignación*".

Para resolver la primera de estas dos cuestiones emplearemos la función **`lp.transport`** del paquete **`lpSolve.`** Los argumentos de esta función son, en este orden:

`cost.mat:` matriz de costos en la que el elemento ij representa el costo de transportar una unidad del origen i al destino j,
`direction:` "`min`" (por defecto) o "`max`", según se trate de minimizar un coste o maximizar un beneficio,
`row.signs:` vector de desigualdades de las restricciones de filas,
`row.rhs:` vector de coeficientes de los lados derechos de las restricciones de filas,
`col.signs:` vector de desigualdades de las restricciones de columnas,
`col.rhs:` vector de coeficientes de los lados derechos de las restricciones de columnas,

integers: vector que representa qué variables deben ser enteras. Si no hay ninguna se pone NULL.

Ejemplo 4: Considérense tres centros de producción A, B y C con una capacidad de producción diaria de 50, 20 y 40 toneladas de carbón, respectivamente. Se quiere programar el transporte diario a cuatro centros de consumo D1, D2, D3 y D4 cuyas necesidades diarias son 35, 35, 22 y 18 toneladas respectivamente. Se conoce la matriz de costes unitarios desde cada origen a cada punto de destino:

	D1	D2	D3	D4
A	10	30	15	8
B	12	25	5	35
C	20	7	14	22

Organizar el transporte que haga mínimo el coste diario; es decir, determinar las cantidades diarias que hay que trasladar de cada centro de producción a cada centro de consumo.

```
> library(lpSolve)
> A<-lp.transport(cost.mat=matrix(c(10,30,15,8,12,25,5,35,20,7,14,22),nrow=3,byrow=T),
+ direction="min",row.signs=c(rep("=",3)),row.rhs=c(50,20,40),
+ col.signs=c(rep("=",4)),col.rhs=c(35,35,22,18),integers=NULL)

> A
Success: the objective function is 897
> A$solution
     [,1] [,2] [,3] [,4]
[1,]   32    0    0   18
[2,]    0    0   20    0
[3,]    3   35    2    0
```

El resultado se interpreta del siguiente modo: hay que llevar 32t de carbón del centro de producción A a D1, 18t de A a D4, 20t de B a D3, 3t de C a D1, 35t de C a D2 y 2t de C a D3.

Para resolver un problema de asignación utilizaremos la función **lp.assign** del paquete **lpSolve.** Los argumentos de esta función son, en este orden:

cost.mat: matriz de costos en la que el elemento ij representa el costo o beneficio de asignar el origen i al destino j,
direction: "min" (por defecto) o "max", según se trate de minimizar un coste o de maximizar un beneficio.

Ejemplo 5: Supongamos que la tabla que se presenta a continuación representa lo siguiente: A, B, C y D son cuatro abogados de un cierto bufete y M1, M2, M3 y M4 son 4 casos a resolver. Los valores de la tabla son los tiempos estimados en horas que emplearían los distintos abogados con cada uno de los casos. Determinar la forma óptima de asignar los casos a los abogados de forma que cada uno aborde un caso diferente y que el tiempo total empleado sea mínimo.

	M1	M2	M3	M4
A	34	10	15	28
B	16	15	22	12
C	10	25	13	20
D	30	19	27	31

```
> library(lpSolve)
>                                              A>
lp.assign(cost.mat=matrix(c(34,10,15,28,16,15,22,1
2,10,25,13,20,30,19,27,31),nrow=4,byrow=T),
+ direction="min")
> A
Success: the objective function is 56
> A$solution
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    0
[2,]    0    0    0    1
[3,]    1    0    0    0
[4,]    0    1    0    0
```

Por lo tanto, el abogado A será asignado al caso M3, el abogado B al caso M4, C al caso M1 y D al caso M2.

4.- OTRAS OPCIONES PARA RESOLVER PROBLEMAS DE PROGRAMACIÓN LINEAL EN *R*

Una forma alternativa de abordar el problema de programación lineal en *R* es utilizando la función **linp** del paquete **limSolve**. Resolvamos el ejemplo 1 con esta función:

```
> library(limSolve)
> a<-c(-60,-35,-20) #Como la función linp se
aplica sólo a casos de minimización, multiplicamos
por -1 los coeficientes de la función objetivo
> coef.men.ig<-c(8,6,1,4,2,1.5,2,1.5,0.5,0,1,0)
> coef.may.ig<-coef.men.ig*(-1) # Como la función
linp exige que todas las restricciones sean del
tipo >=, debemos multiplicar por -1 los
coeficientes de los lados derechos de las
restricciones del tipo <=
> A1<-matrix(coef.may.ig,nrow=4,byrow=T)
```

```
> A1
     [,1] [,2] [,3]
[1,]   -8 -6.0 -1.0
[2,]   -4 -2.0 -1.5
[3,]   -2 -1.5 -0.5
[4,]    0 -1.0  0.0
> b1<-c(-48,-20,-8,-5) #Igual que antes debemos multiplicar por -1 los lados derechos de las restricciones, pues sólo se admiten restricciones del tipo >=

> linp(E=NULL,F=NULL,G=A1,H=b1,Cost=a) #E y F son los lados izquierdo y derecho de las restricciones de igualdad que, en este caso, no hay
$X
[1]  0.0  1.6 11.2

$residualNorm
[1] 0

$solutionNorm
[1] -280

$IsError
[1] FALSE

$type
[1] "simplex"
```

Por último, se ha de señalar que hay otras opciones, entre las que podríamos destacar:

- Utilizar la función **`solveLP`** del paquete **`linprog`**.

- Cargar el paquete **`glpk`** que proporciona una interface para el programa GLPK (GNU Linear Programming Package), que es un software libre para resolver problemas de programación lineal y de programación lineal entera mixta de gran escala y problemas relacionados.

- Utilizar el paquete **`rcdd`** para resolver problemas PL con aritmética exacta mediante la función **`lpcdd`**.