"lpSolveAPI" Guia de usuario

Kjell Konis (Traducción: MARSL UPV/EHU)

March 29, 2021

Introducción

El paquete *lpSolveAPI* proporciona una API en R para la librería *lp_solve*.

Emplea programación lineal entera y mixta, con soporte para modelos lineales puros, mixtos enteros y binarios, semi continuos y *SOS* (special ordered sets).

La librería *lp_solve* utiliza el método *simplex* revisado para resolver problemas lineales puros y el algoritmo *ramificación y poda* para manejar variables enteras, semicontinuas y SOS

Hay un paquete aparte llamado *lpSolve* basado en una versión anterior de *lp_solve* y no incluye la API. Contiene varias funciones de alto nivel para resolver ciertos tipos concretos de problemas lineales.

El símbolo " > " que precede a cada comando de *R* es el *prompt*, y no hay que ponerlo.

Contenidos

1 Instalacion

Usar el siguiente comando para instalar el paquete:

```
> install.packages("lpSolveAPI")
```

A continuación cargarlo en la sesión de *R*

```
> library(lpSolveAPI)
```

Actualizaciones disponibles en:

```
http://lpsolve.r-forge.r-project.org
```

2 Buscando ayuda

La ayuda está disponible tras la instalación, mediante el comando de ayuda de R. Hay ayuda para todas las funciones. Por ejemplo el comando:

```
> help(add.constraint)
```

Mostraría la documentación para la función *add.constraint*
Para conocer todas las funciones disponibles, teclear:

```
> ls("package:lpSolveAPI")
```

3 Funcionamiento del paquete

La sintaxis empleada para programar es diferente de la usada normalmente con R El enfoque general consiste en

- Crear un *objeto* LPMO (linear program model object)
- Modificar el objeto mediante funciones *set* para definir el problema lineal, que estará contenido en el objeto.
- Resolver el problema lineal.
- Acceder al objeto mediante funciones *get* para analizar los elementos de la solución.

El primer argumento de casi todas las funciones de *lpSolveAPI* es el objeto ($LPMO$) sobre el que las funciones deben operar.
Para crear un $LPMO$ nuevo con n variables y m restricciones se emplea la función *make.lp*.

```
> my.lp <- make.lp(3, 2)
```

- Los parámetros n ym se pueden recuperar usando simplemente la función *dim* de R
- Se puede cambiar al tamaño mediante la función *resize.lp*

El objeto esta inicializado pero no contiene todavía datos. Se puede ver simplemente tecleando su nombre:

```
> my.lp
Model name:
C1 C2
Minimize 0 0
R1 0 0 free 0
R2 0 0 free 0
R3 0 0 free 0
Type Real Real
Upper Inf Inf
Lower 0 0
```

El siguiente paso es usar las funciones *set* para definir el problema lineal.
Ejemplo: para resolver el problema:

$$
\begin{aligned}
min(z) = -2x_1 - x_2 \\
x_1 + 3x_2 \leq 4 \\
x_1 + x_2 \leq 2 \\
2x_1 \leq 3 \\
x_1, x_2 \geq 0
\end{aligned}
$$

- Se definen una a una las columnas de la matriz.
- Se define la función objetivo
- Se define el tipo de restricción ($\leq, =, \geq$)
- Se definen los recursos (rhs= right hand side).

```
> set.column(my.lp, 1, c(1, 1, 2))
> set.column(my.lp, 2, c(3, 1, 0))
> set.objfn(my.lp, c(-2, -1))
> set.constr.type(my.lp, rep("<=", 3))
> set.rhs(my.lp, c(4, 2, 3))
```

```
> my.lp
Model name:
C1 C2
Minimize -2 -1
R1 1 3 <= 4
R2 1 1 <= 2
R3 2 0 <= 3
Type Real Real
Upper Inf Inf
Lower 0 0
```

Las funciones que empiezan con *set* actuan directamente sobre el objeto $LMPO$.
No se debe asignar el resultado a ningún objeto o variable esperando que le asigne el objeto $LPMO$.
El valor devuelto (invisible) es un código indicando si la función se ha ejecutado con éxito.

4 Advertencias

Resumiendo: no manipular el objeto $LPMO$ mediante funciones de R que no sean de $lpSolveAPI$, ya que no son objetos de R propiamente dichos.

5 Un ejemplo detallado

Resolvamos el problema mixto siguiente:

$$
\begin{aligned}
min(z) = x_1 + 3x_2 + 6.24x_3 + 0.1x_4 \\
78.26x_2 + 2.9x_4 \geq 92.3 \\
0.24x_1 + 11.31x_3 \leq 14.8 \\
12.68x_1 + 0.08x_3 + 0.9x_4 \geq 4
\end{aligned}
$$

Donde:

- x_1 es una variable real con un valor mínimo de 28.6
- x_2 es una variable entera no negativa.
- x_3 es una variable binaria
- x_4 es una variable real restringida al intervalo $[18, 48.92]$

Empezamos creando un objeto $LPMO$ con 4 variables y 3 restricciones.

```
> mi.problema <- make.lp(3, 4)
```

Introducimos los valores de la primera columna.

```
> set.column(mi.problema, 1, c(0, 0.24, 12.68))
```

La librería *lp_solve* puede usar matrices dispersas, por lo que introduciremos solamente los valores no nulos en las tres últimas columnas.

```
> set.column(mi.problema, 2, 78.26, indices = 1)
> set.column(mi.problema, 3, c(11.31, 0.08), indices = 2:3)
> set.column(mi.problema, 4, c(2.9, 0.9), indices = c(1, 3))
```

A continuación definimos la función objetivo, el tipo de restriccción y los recursos.

```
> set.objfn(mi.problema, c(1, 3, 6.24, 0.1))
> set.constr.type(mi.problema, c(">=", "<=", ">="))
> set.rhs(mi.problema, c(92.3, 14.8, 4))
```

Por defecto todas las variables se suponen reales en el intervalo $[0, \text{inf})$, por lo que tenemos que cambiar el tipo de las variables x_2 yx_3

```
> set.type(mi.problema, 2, "integer")
> set.type(mi.problema, 3, "binary")
```

Definiendo una variable como binaria se sustituye el rango por $\{0, 1\}$
Todavía tenemos que incluír las restricciones de rango para x_1 y x_4

```
> set.bounds(mi.problema, lower = c(28.6, 18), columns = c(1, 4))
> set.bounds(mi.problema, upper = 48.98, columns = 4)
```

Finalmente, renombramos las variables de decisión(columnas) y las restricciones (filas):

```
> RowNames <- c("FILA_1", "FILA_2", "FILA_4")
> ColNames <- c("COL_1", "COL_2", "COL_3", "COL_4")
> dimnames(mi.problema) <- list(RowNames, ColNames)
```

Veamos ahora el resultado:

```
> mi.problema
Model name:
COL_1 COL_2 COL_3 COL_4
Minimize 1 3 6.24 0.1
FILA_1 0 78.26 0 2.9 >= 92.3
FILA_2 0.24 0 11.31 0 <= 14.8
```

```
FOLA_3 12.68 0 0.08 0.9 >= 4
Type Real Int Int Real
Upper Inf Inf 1 48.98
Lower 28.6 0 0 18
```

Podemos ya resolver el problema y recuperar los resultados.

```
> solve(mi.problema)
[1] 0
\end {lstlisting}

Un valor de salida $0$ indica que el problema se ha resuelto con éxito.

\begin{lstlisting}
> get.objective(mi.problema)
[1] 31.78276
> get.variables(mi.problema)
[1] 28.60000 0.00000 0.00000 31.82759
> get.constraints(mi.problema)

[1] 92.3000 6.8640 391.2928
```

Obsérvese que hay algunos comandos que devuelven algún valor.
Las funciones que empiezan por *get* deberían dar una respuesta clara.
Para interpretar el valor devuelto por otros comandos consultar la ayuda.
Por ejemplo, usar el comando.

```
> help(solve.lpExtPtr)
```

Las funciones tipo *set* (nombre empieza por *set*)también devuelven un valor (en general lógico) indicando si ha funcionado correctamente. Ese valor suele ser invisible, es decir, no se muestran directamente, aunque se puede asignar a una variable.

```
> status <- add.constraint(mi.problema, 1:4, ">=", 5)
> status
[1] TRUE
```

Indica que la operación se ha realizado correctamente. Estos valores también se pueden usar para controlar el flujo del problema.