lpSolveAPI Package Users Guide

Kjell Konis

Contents

1 Introduction

Note that there is also an lpSolve package available on CRAN. The CRAN version of the package is based on lp_solve version 5.5.0.8 and does not include the API. Note

The > shown before each R command is the R prompt. Only the text after > must be entered.

1.1 Installation

Use the following command to install the package

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

then load it into your R session.

```
> library(lpSolveAPI)
```

1.2 Getting Help

Documentation is provided for each function in the lpSolve package using R's built-in help system. For example, the command

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

will display the documentation for the add.constraint function.

1.3 Conventions used in the lpSolveAPI Package

The syntax used to program lp_solve via the lpSolveAPI package is different from that normally used in R. The general approach used in the lpSolveAPI package is to create an *lpSolve linear program model object* (LPMO), use the set accessor methods to define the linear program in that object, solve the linear program and finally use the get accessor methods to retrieve elements of the solution. The first argument to almost all of the functions in the lpSolveAPI package is therefore the lpSolve linear program model object that the function is meant to operate on.

A new lpSolve linear program model object with m constraints and n decision variables can be created using the `make.lp` function. For example, the following command creates an lpSolve linear program model object with 3 constraints and 2 decision variables.

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

The number of constraints m and the number of decision variables n can be retrieved using the generic `dim` function. Note that the objective function and right-hand-side are not counted in the dimension of the LPMO. Assignment via the `dim` function is not supported; to change the dimension of an LPMO it is necessary to use the `resize.lp` function.

The LPMO is now initialized but contains no data.

```
> 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
```

The next step is to use the set accessor methods to define a linear programming problem. Suppose we wish to solve the following linear program

$$\begin{array}{rrcrcl} \text{minimize:} & -2x_1 & - & x_2 & & \\ \text{subject to:} & x_1 & + & 3x_2 & \leq & 4 \\ & x_1 & + & x_2 & \leq & 2 \\ & 2x_1 & + & & \leq & 3 \end{array}$$

with $x_1 \geq 0$ and $x_2 \geq 0$. The best way to build a model in lp_solve is columnwise; that is to set the columns then the objective function, constraint types and right-hand-side. For example,

```
> 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
```

The set accessor methods (the functions with names beginning with set) operate directly on a lpSolve linear program model object. Do not assign the output of a set accessor method and expect an lpSolve linear program model object. The (invisible) return value of these functions is generally a logical status code indicating whether the function executed sucessfully.

1.4 Caveats

The lpSolveAPI package provides an API for building and solving linear programs that mimics the lp_solve C API. This approach allows much greater flexibility but also has a few caveats. The most important is that the lpSolve linear program model objects created by make.lp and read.lp are not actually R objects but external pointers to lp_solve 'lprec' structures. R does not know how to deal with these structures. In particular, R cannot duplicate them. Thus you must never assign an existing lpSolve linear program model object in R code.

Consider the following example. First we create an empty model `x`.

```
> x <- make.lp(2, 2)
```

Then we assign x to y.

```
> y <- x
```

Next we set some columns in x.

```
> set.column(x, 1, c(1, 2))
> set.column(x, 2, c(3, 4))
```

And finally, take a look at y.

```
> y
```

```
Model name:
            C1    C2
Minimize     0     0
R1           1     3  free  0
R2           2     4  free  0
Type      Real  Real
Upper      Inf   Inf
Lower        0     0
```

The changes we made in `x` appear in `y` as well. Although `x` and `y` are two distinct objects in R, they both refer to the same lp_solve 'lprec' structure.

1.5 Learning by Example

```
> lprec <- make.lp(0, 4)
> set.objfn(lprec, c(1, 3, 6.24, 0.1))
> add.constraint(lprec, c(0, 78.26, 0, 2.9), ">=", 92.3)
> add.constraint(lprec, c(0.24, 0, 11.31, 0), "<=", 14.8)
> add.constraint(lprec, c(12.68, 0, 0.08, 0.9), ">=", 4)
> set.bounds(lprec, lower = c(28.6, 18), columns = c(1, 4))
> set.bounds(lprec, upper = 48.98, columns = 4)
> RowNames <- c("THISROW", "THATROW", "LASTROW")
> ColNames <- c("COLONE", "COLTWO", "COLTHREE", "COLFOUR")
> dimnames(lprec) <- list(RowNames, ColNames)
```

Lets take a look at what we have done so far.

```
> lprec
```

```
Model name:
            COLONE    COLTWO  COLTHREE   COLFOUR
```

```
Minimize         1         3      6.24       0.1
THISROW          0     78.26         0       2.9  >=  92.3
THATROW       0.24         0     11.31         0  <=  14.8
LASTROW      12.68         0      0.08       0.9  >=     4
Type          Real      Real      Real      Real
Upper          Inf       Inf       Inf     48.98
Lower         28.6         0         0        18
```

Now lets solve the model.

```
> solve(lprec)
```

```
[1] 0
```

```
> get.objective(lprec)
```

```
[1] 31.78276
```

```
> get.variables(lprec)
```

```
[1] 28.60000  0.00000  0.00000 31.82759
```

```
> get.constraints(lprec)
```

```
[1]  92.3000   6.8640 391.2928
```

Note that there are some commands that return an answer. For the accessor functions (generally named get.*) the output should be clear. For other functions (e.g., solve), the interpretation of the returned value is described in the documentation. Since solve is generic in R, use the command

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

to view the appropriate documentation. The assignment functions (generally named `set.*`) also have a return value - often a logical value indicating whether the command was successful - that is returned invisibly. Invisible values can be assigned but are not echoed to the console. For example,

```
> status <- add.constraint(lprec, c(12.68, 0, 0.08, 0.9), ">=",
+     4)
> status
```

```
[1] TRUE
```

indicates that the operation was successful. Invisible values can also be used in flow control.

2 Further Examples

2.1 Solving Dense Mixed Integer/Linear Programs

2.2 Sparse Linear Programs: The Transportation Problem

2.3 Integer Variables: Non-Integer c_{ij}

2.4 Binary Variables: The 8 Queens Problem