glpkAPI – Quick Start

Gabriel Gelius-Dietrich

February 13, 2012

1 Introduction

The package *glpkAPI* provides a low level interface to the C API of GLPK[1], the GNU Linear Programming Kit. It is similar in purpose to the package *glpk*[2], but *glpkAPI* relies on a separate installation of GLPK.

2 Installation

The package *glpkAPI* depends on a working installation of GLPK (in particular libraries and header files). It is recommended to link GLPK to the GNU Multiple Precision Arithmetic Library Library (GMP)[3] in order to gain more performance when using the exact simplex algorithm. See `INSTALL` for installation instructions and platform specific details.

3 Usage

3.1 Creating and solving a linear optimization problem

In the following, an example lp-problem will be created and solved. It is the same lp-problem which is used in the GLPK manual:

maximize

$$z = 10x_1 + 6x_2 + 4x_3$$

subject to

$$\begin{aligned} x_1 + \ x_2 + \ x_3 &\leq 100 \\ 10x_1 + 4x_2 + 5x_3 &\leq 600 \\ 2x_1 + 2x_2 + 6x_3 &\leq 300 \end{aligned}$$

where all variables are non-negative

$$x_1 \geq 0, \ x_2 \geq 0, \ x_3 \geq 0$$

1 Andrew Makhorin: GNU Linear Programming Kit, Version 4.42 (or higher) `http://www.gnu.org/software/glpk/glpk.html`

2 Maintained by Lopaka Lee. Available on CRAN `http://cran.r-project.org/`

3 `http://gmplib.org/`

Load the library.

```
> library(glpkAPI)
```

Create an empty problem object.

```
> prob <- initProbGLPK()
```

Assign a name to the problem object.

```
> setProbNameGLPK(prob, "sample")
```

Set the direction of optimization. The object `GLP_MAX` is a predefined constant used by GLPK. A list of all available contants is written in the documentation `glpkConstants`.

```
> setObjDirGLPK(prob, GLP_MAX)
```

Add three rows and three columns to the problem object.

```
> addRowsGLPK(prob, 3)

[1] 1

> addColsGLPK(prob, 3)

[1] 1
```

Set row and column names.

```
> setRowNameGLPK(prob, 1, "p")
> setRowNameGLPK(prob, 2, "q")
> setRowNameGLPK(prob, 3, "r")
> setColNameGLPK(prob, 1, "x1")
> setColNameGLPK(prob, 2, "x2")
> setColNameGLPK(prob, 3, "x3")
```

Set the type and bounds of the rows.

```
> setRowBndGLPK(prob, 1, GLP_UP, 0, 100)
> setRowBndGLPK(prob, 2, GLP_UP, 0, 600)
> setRowBndGLPK(prob, 3, GLP_UP, 0, 300)
```

Set the type and bounds of rows using a function which has the ability to work with vectors.

```
> lb <- c(0, 0, 0)
> ub <- c(100, 600, 300)
> type <- rep(GLP_UP, 3)
> setRowsBndsGLPK(prob, 1:3, lb, ub, type)
```

Set the type and bounds of the columns.

```
> setColBndGLPK(prob, 1, GLP_LO, 0, 0)
> setColBndGLPK(prob, 2, GLP_LO, 0, 0)
> setColBndGLPK(prob, 3, GLP_LO, 0, 0)
```

Set the objective function.

```
> setObjCoefGLPK(prob, 1, 10)
> setObjCoefGLPK(prob, 2, 6)
> setObjCoefGLPK(prob, 3, 4)
```

Set the type and bounds of columns and the objective function using a function which has the ability to work with vectors.

```
> lb <- c(0, 0, 0)
> ub <- lb
> type <- rep(GLP_LO, 3)
> obj <- c(10, 6, 4)
> setColsBndsObjCoefsGLPK(prob, 1:3, lb, ub, obj, type)
```

Load the constraint matrix.

```
> ia <- c(1, 1, 1, 2, 3, 2, 3, 2, 3)
> ja <- c(1, 2, 3, 1, 1, 2, 2, 3, 3)
> ar <- c(1, 1, 1, 10, 2, 4, 2, 5, 6)
> loadMatrixGLPK(prob, 9, ia, ja, ar)
```

Solve the problem using the simplex algorithm.

```
> solveSimplexGLPK(prob)

[1] 0
```

Retrieve the value of the objective function after optimization.

```
> getObjValGLPK(prob)

[1] 733.3333
```

Retrieve the values of the structural variables (columns) after optimization.

```
> getColPrimGLPK(prob, 1)

[1] 33.33333

> getColPrimGLPK(prob, 2)

[1] 66.66667
```

```
> getColPrimGLPK(prob, 3)

[1] 0
```

Retrieve all primal values of the structural variables (columns) after optimization.

```
> getColsPrimGLPK(prob)

[1] 33.33333 66.66667  0.00000
```

Retrieve all dual values of the structural variables (columns) after optimization (reduced costs).

```
> getColsDualGLPK(prob)

[1]  0.000000  0.000000 -2.666667
```

Print the solution to text file `sol.txt`.

```
> printSolGLPK(prob, "sol.txt")

[1] 0
```

Write the problem to file `prob.lp` in lp format.

```
> writeLPGLPK(prob, "prob.lp")

[1] 0
```

Read problem from file `prob.lp` in lp format.

```
> lp <- initProbGLPK()
> readLPGLPK(lp, "prob.lp")

[1] 0
```

Free memory, allacated to the problem object.

```
> delProbGLPK(prob)
> delProbGLPK(lp)
```

3.2 Setting control prarmeters

All parameters and possible values are described in the documentation, see

```
> help(glpkConstants)
```

for details. The control parameters used by *glpkAPI* have the same names like those from GLPK, except that they are written in capital letters. For example, the parameter `tm_lim` in GLPK is `TM_LIM` in *glpkAPI*. The prarmeters are stored in a structure available only once per R session. Set the searching time limit to one second.

```
> setSimplexParmGLPK(TM_LIM, 1000)
```

4 Finding function names

The function names in *glpkAPI* are different from the names in GLPK, e. g. the function `addColsGLPK` in *glpkAPI* is called `glp_add_cols` in GLPK. The directory `inst/` containes a file `c2r.map` which maps a GLPK function name to the corresponding *glpkAPI* function name. Additionally, all man-pages contain an alias to the GLPK function name. The call

```
> ?glp_add_cols
```

will bring up the man-page of `addColsGLPK`. Keep in mind that some of the GLPK functions do not work on vectors. For example the function `setColBndGLPK` (which is `glp_set_col_bnds` in GLPK) sets the upper and lower bounds for exactly one column. The function `setColsBndsGLPK` in *glpkAPI* can handle a vector of column indices.

Assume, we have a problem containing 1000 columns and 600 rows, with all variables having a lower bound of zero and an upper bound of 25. The problem will be created as follows.

```
> prob <- initProbGLPK()
> addColsGLPK(prob, 1000)

[1] 1

> addRowsGLPK(prob, 600)

[1] 1
```

Now we can set the column bounds via `mapply` and `setColBndGLPK`.

```
> system.time(
+    mapply(setColBndGLPK, j = 1:1000,
+    MoreArgs = list(lp = prob, type = GLP_DB, lb = 0, ub = 25))
+ )

   user  system elapsed
  0.031   0.003   0.034
```

Or we use the simpler call to `setColsBndsGLPK`.

```
> system.time(
+    setColsBndsGLPK(prob, j = 1:1000,
+                    type = rep(GLP_DB, 1000),
+                    lb = rep(0, 1000),
+                    ub = rep(0, 1000))
+ )

   user  system elapsed
  0.001   0.000   0.000
```

The latter call is also much faster.