Need for speed with R

Michail Tsagris and Manos Papadakis

When it comes to large scale data and or big scale simulations, researchers tend to rely on matlab or even better C++ for faster and more efficient programming. MCMC is a classic example of why researchers tend to use C++. But not everybody programs in C++ and not everybody has access to matlab due to its license fee. Biologists for example, or bioinformaticians like to use R and in fact R has many relevant (and fast) packages.

On the other hand, we have entered the "big data" era, where large scale datasets are available. Bear in mind that the computational burden goes hand in hand with the size of the data. In fact, the relationship between them is super-linear. Hence the need to handle, manipulate and perform statistical analysis with large scale data efficiently is evident.

Are there R packages efficiently written? The answer is yes, "glmnet", "speedglm", "matrixStats", "geneFilter" (Bioconductor), "quantreg" to name a few are packages which offer really fast functions. Are there more? Yes of course. Which are they, and what do they offer? The answer is not easy and depends upon the field of statistics and or Bioinformatics. Do they cover all areas? No. There are many more packages offering very fast functions, but one must search a lot and compare the speed of the functions before deciding which package to use.

The obvious solution to the above would be to create a package which depends upon tens of other packages, importing one or more functions from each package. Alternatively, a package, or better a collection of functions, really *fast* or *faster than* the available ones, all in one place. This package started is *Rfast*.

Our goal is to implement many statistical (mainly), linear algebra, and other utility, functions. We will work on case by case base for many methods. For example, in mixed effects model, there are balanced and unbalanced designs, MLE or REML, presence or absence of covariates. These cases will be treated differently in order to minimize the computational time. Below is a flavor of our portfolio of functions.

- MLE of univariate and multivariate distributions.
- Regression models, with many covariates examining their significance independently.
- Regression models for large scale data, with many tens of thousands of observations and hundreds of predictor variables.
- Many hypothesis testing procedures, parametric and non parametric, for one, two or more samples.
- Linear algebra related functions, operations with matrices, etc.
- Design of experiments, ANCOVA, two way ANOVAs, split-plot designs, etc.
- Mixed models, random effects with MLE, REML, regression etc.
- Algorithms, such as the PC algorithm (for network construction) and the K-NN algorithm.
- Utility functions, such as combinatorics, lower and upper triangular matrix, diagonals, etc.

The functions in Rfast are of general use, not promoting our work specifically and very importantly, they will be fast, even on old computers or computers with low computational power. To give an example of this, let us assume that there are two friends. Alex, who has an old laptop and Ritchie, who has a newly bought desktop. Both of them, using the same R version download the same R package and perform a heavy statistical analysis Alex requires 45 seconds, whereas Ritchie requires only 10. The obvious conclusion is that Alex should buy a new computer. The next day, Alex presents his new results to Ritchie and they require only 0.45 seconds (in his old laptop!!!). The new conclusion is that Ritchie should buy an old laptop (!)[1]. The moral behind is the obvious, what everybody agrees and says, but not everybody does. It is not a matter of the computer, but of the implementation. And this is exactly what we are doing. To continue, the story, this function is "colvarcomps.mle", which after removing the "for" loop became 70 times faster (than the 0.45 seconds version), a total oof **7,000 (!!!) times**.

We have explored many R packages and built-in R functions and to the best of our knowledge, our functions are indeed faster. In addition, we do keep looking back and searching for alternative functions, better use of mathematics, or ways to improve our functions in terms of computational efficiency. An example of this is the column-wise ANOVA tests with unequal variances (*ftests* function) as can be seen in the examples below.

As for testing our functions, there are multiple directions. In terms of accuracy, the simplest thing to do is test the results versus the built-in or packages implemented functions. In the logistic regression case, for example, the Newton-Raphson algorithm is used. We carefully implemented the mathematics, both in R and C++ and tested both of them against R's "glm" command. Other functions use C++ commands, such as sorting or finding the n-th element of a vector. We test even them against R's built in functions. For every self-written functions, we make sure the mathematics is strictly followed. In addition, we run our functions against the ones in R or other R packages a few thousands of times as an extra step of testing.

One limitation, we currently have, for the sake of speed is that most of the functions allow matrices or vectors. The reason for this is that we mainly work with numerical data. The G^2 test could be seen as an exception to this, but even this test accepts numerical matrices only. In the linear regressions case ("regression" command) we accept data.frames and we plan to allow this object for other regressions as well. Furthermore, we avoid checks, a column with zero variance for example (this is a function to be added), or wrong inputs. We leave these to the user. As for the NA cases, currently only a few functions handle them, but in the future we will work in this direction as well. Our target is to have bare bones functions which work faster (or a lot faster) than the present ones.

Let us now give a few examples ran in an old Dell computer (Intel Core 2 Duo CPU, E8400 at 3GHz, with 4 GB RAM and 64 bit Windows 10 operating system). The duration varies according to the computer, so we also provide the code to run the examples yourself. The key message is the speed up factor which can be from 1.5 up to 250, depending on the function.

[1] *The above story is totally true; it happened with the first author.*

G^2 test of indepence for categorical data

```
nvalues<- 3     ;     nvars<- 100     ;     nsamples<- 10000
data<- matrix( sample( 0:(nvalues - 1), nvars * nsamples, replace = TRUE ), nsamples, nvars )
dc<- rep(nvalues, nvars)
ela<- function(data) {
  p <- dim(data)[2]
for (i in 1:(p-1)) {
   for (j in (i + 1):p ) {chisq.test(data[, i], data[, j])    }  }  }
```

system.time(g2Test_univariate(data = data, dc = dc))   ### 0.17 - 0.18 seconds

system.time(ela(data)) ## 48-49 seconds

Column-wise F-tests

```
x <- matrix(rnorm(500 * 10000), ncol = 10000)
ina <- rbinom(500, 3, 0.5) + 1       ;       ina2 <- as.factor(ina)
microbenchmark(genefilter::colFtests(x, ina2), ftests(x, ina), times = 50)
```

colFtests: average time 999 miliseconds

ftests: average time 100 miliseconds (current version)

Univariate regressions with continuous predictors

```
y <- rnorm(1000)

x <- matrix(rnorm(1000 * 10000), ncol = 10000)

ela2 <- function(y, x) {

  p <- dim(x)[2]

for (i in 1:p) lm.fit( cbind(1, x[,i]), y )  }
```

system.time(univglms(y, x))   ## 0.10 seconds

system.time(ela2(y, x))   ## 2.57 seconds