Build R package to Run Simulation in HPC

Sunthud Pornprasertmanit

June 19, 2015

In this class, I will show you the method that I use to run simulation lately. This method is the most efficient way to run simulation in my opinion. There are two things in addition to the previous class. The first thing is how to separate the jobs in HPC. We can make each job representing each combination of design conditions such that all replications in each combination are run in the same jobs. Alternatively, we can make each job representing each replication. That is, each job will go through all combinations of design conditions once. I found that the latter method is more efficient because the usage time across jobs is stable. We can expect the finishing time. Also, we can control each job to not take time over the HPC limit, which is three days.

Secondly, I currently build a package to save functions that run all combinations of design conditions, distribute files, or save the L'Ecuyer seed. Building R package for simulation has many advantages. First, building package process will check our codes briefly. Some bugs can be caught before going to HPC. Second, building package requires users to build documentation. When I do not use it for a while, I can come back and catch up easier. Third, it makes me communicate with my coauthors easier. When I need my coauthors to run a simulation, coauthors can help me easily.

1 Simulation example

This section will cover an simulation example that divide jobs based on different replications. Let's make a simulation to compare bootstrap confidence interval, profile-likelihood confidence interval, and Wald confidence interval on slope of predictor from simple logistic regression. There are three design conditions here: (a) intercepts (-10, -2, 0, 2, and 10), (b) slopes (0, 1, 2, and 3), and (c) sample sizes (50, 100, 200, 400, and 800).

First, we need the seed number from L'Ecuyer method. To be efficient, we can create the seed numbers for all replications in advance.

```
> set.seed(123321)
> nrep <- 1000
> library(parallel)
> RNGkind("L'Ecuyer-CMRG") # Use L'Ecuyer method
> s <- .Random.seed # Extract the current seed number
> lecuyerseed <- NULL
> for (i in 1:nrep) {
```

```
+   lecuyerseed <- rbind(lecuyerseed, s) # Save seed number to the list
+   s <- nextRNGStream(s) # Build independent seed number
+ }
> save(lecuyerseed, file = "lecuyerseed.Rda")
```

`lecuyerseed` saves the seed number in each row. We will need this matrix of seed number in the built package. We will save this seed number to a file named `"lecuyerseed.Rda"`.

Next, let's try to create a function that generates data from a given combination of design conditions, analyze the generated data, get Wald, profile-likelihood, and bootstrap confidence interval of the slope, and return whether the confidence interval covers the population slopes.

```
> runrep <- function(cond, seednum) {
+   RNGkind("L'Ecuyer-CMRG")
+   assign(".Random.seed", seednum[unlist(cond[4]),], envir = .GlobalEnv)
+   a <- unlist(cond[1])
+   b <- unlist(cond[2])
+   n <- unlist(cond[3])
+   x <- rnorm(n, 0, 1)
+   pred <- a + b * x
+   predprob <- 1/(1+exp(-pred))
+   y <- runif(length(predprob)) < predprob
+   dat <- data.frame(x, y)
+   mylogit <- glm(y ~ x, data = dat, family = "binomial")
+   result <- matrix(NA, 3, 2)
+   result[1,] <- try(confint.default(mylogit)[2,])
+   result[2,] <- try(confint(mylogit)[2,])
+   confint.boot <- function(mydata) {
+     FUN <- function(data, i) {
+       mylogit <- glm(y ~ x, data = data[i,], family = "binomial")
+       coef(mylogit)[2]
+     }
+     library(boot)
+     bootout <- boot(data = mydata, statistic = FUN, R = 2000, stype = "i")
+     boot.ci(boot.out = bootout, conf = 0.95, type = "bca")[["bca"]][1, 4:5]
+   }
+   result[3,] <- try(confint.boot(dat))
+   cover <- (result[,1] < b) & (b < result[,2])
+   return(cover)
+ }

> # Run one condition
> RNGkind("L'Ecuyer-CMRG")
> runrep(c(-2, 1, 200, 50), seednum = lecuyerseed)

[1] TRUE TRUE TRUE
```

```
> # Run all conditions
> a <- c(-10, -2, 0, 2, 10)
> b <- c(0, 1, 2, 3)
> n <- c(50, 100, 200, 400, 800)
> rep <- 50
> conds <- expand.grid(a, b, n, rep)
> conds <- data.frame(t(conds))
> result <- lapply(conds, runrep, seednum = lecuyerseed)
> result <- do.call(rbind, result)
> save(result, file=paste("result50.rda", sep=""))
```

2 Building a package

To get the package, you need three things: (a) functions, (b) documentation of public functions, and (c) data (including seed for each replication). Regarding to the functions, there are two types of function here. The first function is to find the result for each replication and each combination of design conditions. In the previous example, the `runrep` is the first function. The second function is to create R files for each jobs. In this case, if we have 1,000 replications, we will have 1,000 different R files that this function creates. Each R file will run all combinations of design conditions once. This is exactly the code in the run all conditions. However, we need to create the chunk of codes for 1,000 replications with different replication numbers (50 in the example above). We will make the function as a text. The replication number will be coded as `subRep` and we will use the `gsub` function to replace it with appropriate replication numbers.

```
> distributeFile <- function(startNum, endNum) {
+
+ file <- '
+ # The built package is required to get lecuyerseed.
+ library(mypackage)
+ library(parallel)
+ a <- c(-10, -2, 0, 2, 10)
+ b <- c(0, 1, 2, 3)
+ n <- c(50, 100, 200, 400, 800)
+ rep <- subRep
+ conds <- expand.grid(a, b, n, rep)
+ conds <- data.frame(t(conds))
+ result <- mclapply(conds, runrep, seednum = lecuyerseed, mc.cores = 11)
+ result <- do.call(rbind, result)
+ save(result, file=paste("result", subRep, ".rda", sep=""))
+ '
+
+ for(i in startNum:endNum) {
+   file2 <- gsub("subRep", i, file)
```

```
+   write(file2, file=paste("sim", i, ".R", sep=""))
+ }
+ }
```

Currently, we have two functions, `runrep` and `distributeFile`, and one data set, `lecuyerseed`. We want to build a package named `mypackage` based on these two functions and one data set. To facilitate our work, we can simply use the `package.skeleton` function to provide file structure of the package for us.

```
> package.skeleton(list = c("distributeFile", "runrep", "lecuyerseed"),
+         name = "mypackage")
```

Check the current directory, we will see a folder named `mypackage`. Go inside it. You will see three folders and three files. `data` and `R` folders will save our data set and functions. We do not need to bother them anymore. The `man` folder saves the documentation of data and functions. We need to edit the files inside the folder. Three files are `DESCRIPTION`, `NAMESPACE`, and `Read-and-delete-me`. Ignore and delete the last file. The `DESCRIPTION` file should be edited to provide the name, version, date of publication, and description. You will also need the `Depends` field as follows: `Depends: R(>= 3.2), parallel`, which indicates that this package requires R version 3.2 or higher and the parallel package (provided in R by default). If you have functions that use other packages, you need to put the name of packages here. The `NAMESPACE` file shows the public functions. Because we have two public functions, we can change the content of the file to `export(distributeFile, runrep)`. You can see my examples in the provided files.

The last thing is to edit the documentations in the `man` folder. There are four files in the folder. Each file has different fields to fill in. See the R manual for details, `http://cran.r-project.org/doc/manuals/r-release/R-exts.html#Writing-R-documentation-files`. The markup language is LaTeX. Some but not all fields are required. See my example in the provided files. Note that `mypackage-package.Rd` is not required because all details are similar to the `DESCRIPTION` file. You may delete it.

Now, we have everything ready for the package. Go back to the directory that you see the `mypackage` folder. Open commend prompt (Windows) or terminal program (Mac or Linux). Go to the directory and use the following command:

```
R CMD check mypackage
```

This command will check your package whether the codes, documentations, `DESCRIPTION`, and `NAMESPACE` are good. When you do not get any errors, you can build the package in tar ball as follows:

```
R CMD build mypackage
```

Then, you will get the tar ball named `mypackage_1.0.tar.gz`. You can install the built package by using following commands (regardless of OS):

```
> install.packages("mypackage_1.0.tar.gz", repos = NULL, type = "source")
```

In HPC, enter R and install the built package. Run `distributeFile` function. We can use the following submission script to run the simuluation:

```
#!/bin/csh
#$ -S /bin/bash
#$ -cwd
#$ -t 1-10:1
#$ -V
#$ -q normal
#$ -pe fill 12
#$ -P hrothgar
#$ -N logit

/lustre/work/apps/R/bin/R -vanilla -no-save < sim$SGE_TASK_ID.R >
```

3 Exercise

Build a package to run the Monte Carlo simulation on logistic regression from the exercise in the previous class.