Shark simulation tutorial

Jeffrey Durieux and Elise Dusseldorp

04/02/2018

Contents

1 Introduction

In this document, we describe the set-up of a simulation study and the performance of it on a cluster computer (shark). The document is structured as follows:

- Chapter 2: Specification of a simulation study;
- Chapter 3: Preparation of a simulation study;
- Chapter 4: Performance of a simulation study on a cluster computer;
- Chapter 5: Collecting the results from the cluster computer in one large matrix and evaluation of the results by anova;

2 Specification of a simulation study

In a full factorial simulation study, we consider the following 3 steps:

- Step 1: Gauge and design
- Step 2: Statistical Analysis
- Step 3: Evaluation Criteria

2.1 Step 1: Gauge and design

Start with the formulation of:

- the true scenario(s)/model(s) from which you will generate your artificial data, and specify from which distributions you generate the variables that are included in the model.
- specify the design factors (i.e. the factors you are going to vary systematically). In our example, we refer to these factors as:
  - Samp (sample size, having 4 levels: N = 10, 20, 40, and 80);
  - Es (effectsize, having 3 levels: d = 0.2, 0.5 and 0.8)
- number of replications within each cell: k. In our small example, k = 10

For our example, our design contains in total 4 (Samp) by 3 (Es) 12 cells. For each cell we will generate k = 10 data sets.

2.2 Step 2: Statistical Analysis

Specify the type of analysis you want to perform for each dataset (of each cell of the design). Often you want to compare two types of analysis methods (a new method compared to an "old" method). In our example, we refer to these methods as Method_new and Method_old.

2.3 Step 3: Evaluation Criteria

Specify the evaluation criteria which you want to use to compare the two methods. In our example, we refer to the evaluation criterion as Criterion1.

3 Preparation of a simulation study

Write the following functions in `R`:

- a function that generates one data set for each true scenario. The input arguments of this function are the type of true model and the levels of the factors of the simulation design. In our example, we refer to this function as *MySimDataFunction.r*. The output of this function is a simulated data set (SimData);
- One or two functions that analyse the data according to one method. In our example, we refer to this function as *Method_new.r*. The input argument of this function is a dataset (and possibly more input arguments). The output of this function contains the relevant results. In our example, we refer to this as MyAnalysisResult;
- a function that applies the evaluation criterion. In our example, we refer to this function as *MyEvaluationResult.r*. The input of this function is MyAnalysisResult.

4 Performance of a simulation study on a cluster computer

4.1 The main simulation script:

Below you can see the main simulation script called MainSimulationScript.R. This is the script that that contains your whole simulation study. Here, your design is coded, analyses are specified and evaluation criteria are computed. Moreover, saving your output to your directory is specified.

Some usefull and sometimes necessary functions:

- commandArgs()
  - captures supplied command line arguments (see section 4.2 and 4.3)

- expand.grid()
  - usefull functions that creates a data frame from all combinations of factor variables
- set.seed()
  - important for replication of your simulation study
- do.call()
  - neat function to execute a function with a list of supplied arguments

```r
# MySimulationScript.R

# args is a stringvector
args <- commandArgs(TRUE)
args <- as.numeric(args)

RowOfDesign <- args[1]
Replication <- args[2]

### Simulation design example
samp <- c(10, 20, 40, 80)
es <- c(0.2, 0.5, 0.8)

# Design is a data.frame with all possible combinations of the factor levels
# Each row of the design matrix represents a cell of your simulation design

Design <- expand.grid(samp = samp, es = es)

# set a random number seed to be able to replicate the result exactly
set.seed((Replication + 1000)*RowOfDesign)

###Preparation:

# If you use R packages that are not standard:
# Install the relevant R packages, for example:

#install.packages("ica")

#Always use library() to activate the package

#library(ica) #By the way: this is just to illustrate,
#we do not use this package for our example

### Source the relevant R functions of our example
### These functions are available from:
### https://github.com/Github-MS/Shark/tree/master/Scripts

source("/home/durieuxj/SharkTutorial/MyDataGeneration.R")
source("/home/durieuxj/SharkTutorial/Method_new.R")
source("/home/durieuxj/SharkTutorial/Method_old.R")
source("/home/durieuxj/SharkTutorial/MyEvaluation.R")

######### simulation ###########
# Generate data
SimData <- do.call(MyDataGeneration, Design[RowOfDesign, ]  )

# Analyze data set with Method_new
```

```r
tmp <- proc.time()
MyAnalysisResult1 <- Method_new(SimData)

#Analyze data set with Method_old
MyAnalysisResult2 <- Method_old(SimData)
time <- proc.time() - tmp #save the time to run the analyses of one cell of the design.
# Also possible to time both analyses seperately

#Combine relevant results of the analysis by the two methods in a vector (optional)
MyAnalysisResult <- c(MyAnalysisResult1$p.value, MyAnalysisResult2$p.value)

#Evaluate the analysis results of Method_new and Mehod_old
MyResult1 <- MyEvaluation(MyAnalysisResult1)
MyResult2 <- MyEvaluation(MyAnalysisResult2)

#combine the results in a vector:
MyResult <- c(MyResult1, MyResult2)

# save stuff on export
setwd("/exports/fsw/durieuxj/SharkTutorial/")

# Write output (also possible to first save everything in a list object)

#optional to save the data
save(SimData, file =paste("Simdata","Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

#optional to save the full analysis result
save(MyAnalysisResult, file =paste("Analysis","Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

save(MyResult, file =paste("MyResult", "Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))

#optional to save timing of analyses
save(time, file =paste("Time", "Row", RowOfDesign,
      "Rep", Replication ,".Rdata" , sep =""))
```

4.2 The bash script used to run Jobs

Below you can find a bash script (let's call it RunMySimulation.sh) that is used on Shark to run a R-script The bash script 'starts' the R-script 'MainSimulationScript.R' with the command R CMD BATCH and includes some provided arguments: $i and $j. The $ sign indicates that it is a replacement character, with a for loop we will change those values on the fly (see part 4.3). The values provided to these replacement characeters will be collected with the commandArgs() function in our R-script.

```bash

#!/bin/bash
#$ -S /bin/bash
#$ -q all.q
#$ -N sim_1
#$ -cwd
#$ -j Y
#$ -V
#$ -m be
#$ -M YOUREMAIL@fsw.leidenuniv.nl

R CMD BATCH "--args $i $j" MySimulationScript.R
```

4.3 Start your embarrassingly parallel jobs

We have a R-script with simulation code. We also have a bash script RunMySim.sh. But how do we actually start our simulation? We simply use the *qsub* command in a double for loop. Here our jobs are submitted to shark in a parallel fashion.

Remember, for the current example the simulation design consists out of two factors: Sample size (Samp) and effect size (Es). The factors have 4 and 3 levels respectively. So we have a 4x3 factorial design. And we want to replicate this design $k = 10$ times. Thus, we will generate, analyze and evaluate a total of 4 x 3 x 10 = 120 datasets. This can be done as follow:

```
for replication in {1..10}
do
  for rownumber in {1..12}
  do
    qsub -l h_vmem=1G -v i=$rownumber,j=$replication RunMySim.sh
  done
done
```

For this small simulation example, requesting 1G of memory (-l h_vmem=1G) is enough.

Note: in order to use R on shark, you first need to 'load' it. This can simply be done on the command line, type: module load R/3.3.3 (or select another version)

You can submit the double for loop on the command line in Shark. This double for loop will start 10 x 12 = 120 jobs that will run in parallel. Each job has no dependency on each other, meaning that you can run each row of the design completely seperate. This is known as embarrasingly parallel computing and it saves you a lot of comutation time!

5 Collecting the results

After all submitted jobs are done, you should collect all results from your output folder. Below you can find a script that collects the right output and automatically stores the results in a matrix. This matrix can be used for further analyses such as an ANOVA.

Note that collecting the results can be done on Shark. But sometimes it is easier to first download the results to your own device and then collect the final results.

Note that this script uses the package gtools. You first need to install this package.

```
# Collect all results and put it in a matrix

# set directory with results
setwd("~/Desktop/SharkData/")

library(gtools)

## original design
```

```r
samp <- c(10, 20, 40, 80)
es <- c(0.2, 0.5, 0.8)
Design <- expand.grid(samp = samp, es = es)

# pattern matching with grep
files <- dir()
idx <- grep(pattern = "MyResult", files)
files <- files[idx]

# test to see mixedsort() does it's job
scram <- sample(1:120, 120, replace = F)
files <- files[scram]

files <- mixedsort(files)

###
RepIdxList <- list()
for(rep in 1:10){
  RepIdxList[[rep]] <- grep(pattern = paste("Rep",rep,sep = "" ), files )
}

idx_to_remove <- RepIdxList[[1]] %in% RepIdxList[[10]]
RepIdxList[[1]] <- RepIdxList[[1]][!idx_to_remove]

idx <- unlist(RepIdxList)

files <- files[idx]

# rbind design k times and cbind files names (so that you can check whether the right values are taken)

Results <- do.call(what = rbind, args = replicate(10, Design, simplify = F))
Results <- cbind(Results, files)

# the sapply function loads the results and preserves the file name (in the rows).
# This is usefull for checking whether you did everything right

Res <- t( sapply(files, function(x) get(load(x) ) ) )

Results <- cbind(Results, Res)
colnames(Results) <- c("samp", "es", "files", "res1", "res2")

save(Results, file = "AllResults.Rdata")
```

6 Installing R packages on Shark

R script ‘InstallPackages.R’:

```r
install.packages(c("ica","Matrix"), Sys.getenv("R_LIBS_USER"), repos = "http://cran.case.edu")
```

Bash script: InstallPackages.sh

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -q all.q
#$ -N installpackages
#$ -cwd
#$ -j Y
#$ -V
#$ -m be
#$ -M YOUREMAIL@fsw.leidenuniv.nl

R CMD BATCH InstallPackages.R
```

How to run it in Shark: On the commandline type: qsub InstallPackages.sh

Don’t forget to load R first (module load R/3.3.3)