A SIMPLE PROTOCOL FOR SIMULATIONS IN R

ROGER KOENKER

Abstract. It is easy to get sloppy with the organization of simulation exercises and this usually results in painful experiences at the latter stages of projects, and ultimately to the irreproducibility of results. If one could adhere to a simple, consistent protocol I believe many of these problems could be avoided.

I propose the following rules:

1: Always use `set.seed( )`.
2: Never run simulations interactively, always run them from source files.
3: Document the initial environment.
4: Save the final environment.

There are undoubtedly many ways to implement these rules; I will briefly sketch one way that seems to be quite convenient. Suppose that we create a source file called `sim.R` that looks like this:

```
# toy MC experiment to test the Gossett binary response estimator

source("plink.R")
system("hostname")
date()
sessionInfo()

R <- 500
n <- 500
set.seed(1968)
x <- 5*rnorm(n)
dfs <- c(1,2,6)
A <- array(0,c(3,3,R))
for(i in 1:length(dfs)){
        df <- dfs[i]
        print(paste("i = ",i))
        for(j in 1:R){
                y <- (1.0 * x + rt(n,df) > 0)
                f <- pglm(y ~ x,link="Gossett")
                A[,i,j] <- c(f$nuhat,f$nulo,f$nuhi)
                }
        }
```

Version February 13, 2008. This research was partially supported by NSF grant SES-02-40781.

We begin by `source()`ing some functions from the file `plink.R`. These functions and any data residing in `plink.R` will be saved as part of the global environment at the end of the process and will therefore be available for post-mortem analysis. Next we record the name of the machine, the date and some basic information about the version of R and the versions of the packages that we have attached. Then we initialize the seed for the random number generator and get to work. The print statement is used to monitor progress of the job by occasionally peeking in the resulting `sim.Rout` file using, e.g. the shell command `tail`. In more complicated situations it would often be preferable to write a function that encapsulated a single iteration of the simulation.

The file `sim.R` would normally be invoked from the command line with,

```
R CMD BATCH sim.R
```

By so doing we automatically generate two new files: `sim.Rout` which contains a transcript of the executed session, and `.RData` which by default contains a binary version, in standard R format, of the global environment including all functions and data as it existed at the end of the session. It seems prudent to rename this file to something more meaningful. The process can be automated by the following shell script: These four rules produce a pair of new files `sim.Rout` and `sim.Rda`.

```
R CMD BATCH $1.R
mv .RData $1.Rda
chmod -w $1.R
```

This approach enforces a consistent naming convention. Calling the script `Rbatch`, one would simply invoke it with

```
Rbatch sim &
```

This automatically moves the standard output file `.RData` created by the simulation into a more specifically named file `sim.Rda` that can be preserved for posterity. This still doesn't solve the problem that one can overwrite the input file rerun it, and thereby overwrite the input file. Thus, it is safer if the last step in the shell script changes the mode of the input file so that it is read only.

Remarks

(i) The use of `set.seed( )` is essential if one wants to ensure the reproducibility of results. The R random number generators allow one to restart the sequence by simply resetting the seed to the same value that was used previously.

(ii) The batch approach provides an explicit record of what was done. It also, conveniently, provides automatic timing information on the run.

(iii) The `sessionInfo( )` call documents the version of R and any included packages, thereby enabling restoration of the environment used to create the results. Note that prior versions of R and its packages are available from CRAN. This resolves a long-standing problem with proprietary software for which it

was difficult, or even impossible, to restore prior versions of the software to repeat a prior analysis.

(iv) Use of `save.image( )` at the end of the session, which occurs automatically (by default with the `R CMD BATCH` command), ensures a complete record of what was produced by the originating batch file, and can be easily restored using `load( )`. This allows one to separate the relatively time consuming simulation phase of the project from the analysis phase. The latter typically requires relatively little computational effort but may involve considerable fine tuning to produce appropriate tables and graphics. It is advantageous to have separate files for each table and figure, each of which can rely on load to recreate the output of the simulation. For procedures that produce large `.Rda` files one could also add a compression step. The latter can be accomplished automatically by setting,

```
options(save.image.defaults = list(compress=TRUE))
```

either in the `sim.R` file or in the `.Rprofile` file that gets sourced automatically at startup.

(v) The array `A` that is produced by the simulation can usually be easily manipulated using R's `apply` function to produce tables and or figures as appropriate. For tables intended for LaTeX documents there are several very useful tools for semi-automatically generating LaTeX code available from the `Hmisc` package of Frank Harrell. The `quantreg` package also contains a `latex.table` function adapted from an early version of Harrell's function of the same name.

DEPARTMENT OF ECONOMICS, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN