R Coding Standards for PS239T[*]

Fall 2009

1 Introduction

This document is intended to provide guidelines for writing and formating code in PS239T. It is intended to help with the standardization of variable names, formatting, and other coding elements, thus making code easier to write, read, and decipher. In drawing up this document, the author consulted the Google R Style Guide (`http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`) and the unofficial R style guide (`http://www1.maths.lth.se/help/R/RCC/`). Both have good information for thinking about code formatting in R.

The spacing defaults contained here favor users of Emacs + ESS. Some programs (Tinn-R) enable spacing and indenting but do not supply these as defaults. Other, very basic editors (such as the Rgui contained in the OS X version of R) do not support automatic indentation. Readers are advised to not skip the spacing stuff simply because their editor doesn't support it.

Finally, the author emphasizes that there is nothing special about what follows; R will do what you want it to do whether you follow these conventions or not. Think of what follows as an attempt to enforce a social convention, one in which the goal of writing readable code is given weight next to the goal of writing code that executes.

2 File Names

All R code file names should end with `.R`. File names should describe the project addressed with the code they contain. For instance, a file with code for testing demographic factors in the 2004 National Election Survey might be `nes2004Demog.R`. If a single long code file is separated into several smaller files, the file naming should reflect this. Thus if the code for the 2004 NES study was broken into a data formatting file and a data analysis file, the files might be `nes2004DataProc.R` and `nes2004DataAnaly.R`.

[*]DRAFT 28 September 2009

3 Variable naming

For the purposes of most of the code you will encounter in PS239T and subsequent methods courses, R contains three possible object structures: functions, variables, and constants. Those should be named as follows:

- Functions: The function name should include what the function does and any particulars. It should start with a verb. Thus `calc.sqrt` is preferred to `sqrt`. Either period-separated or camel-type names (i.e., `do.this` or `doThis`) are acceptable.

- Variables: Variable naming is divided into short- and long-format names. Short-format names are appropriate for variables that store data such as counts, indices, or other low-information data. Long-variable names are appropriate for data frames, vectors, and other data objects that are high-information.

  – Short: Use of single lowercase letters (i.e. `n`, `i`) is preferred. Some variables have commonly understood uses: `n` and `m` are usually used for counts; `i`, `j`, and `k` are usually used for the counter in loops, or for holding index values for vectors, matrices, or data frames. Single-letter names such as `c` and `t` are discouraged because they are already function names for standard functions in R. Numerical increments (`n2`, `i3`, etc) are discouraged unless it is very clear from the context what they mean.

  – Long: Variable names should describe the included data. This can either describe the dataset (i.e. `borneo.demog` for a dataset containing demographic data on Borneo) or the specific function of the variable in the code (i.e. `ctl.count` for a variable that contains the count of subjects in an experimental control group). Certain abbreviations are conventional, such as the precursor `n` to indicate the variable holds a count for use in a particular purpose. For instance, you will often see `nSims` for holding the number of simulations to be used in a bootstrap. Period-separated or camel formats are acceptable. Generic variable names (`data`, `x`, etc) are discouraged.

- Constants: You should avoid the use of constants in your code if at all possible. Constants can make your code breakable. Nevertheless, sometimes they are appropriate. Constants should be named consistent with the convention for naming long variables, but should be proceeded by `k`. That is, for a variable that holds the Planck constant[1], the variable could be `kPlanck`.

[1]A value in quantum mechanics relating the energy of light to its frequency, as in $E = h\nu$, of the value 6.0636×10^{-34} J$\cdot$ s

4 White space and indenting

Use of appropriate white space and indenting will make your function much easier to read. For instance, the following is very difficult to read:

```
for(i in 1:length(dim.mat)){mat.i <- matrix(nrow=10, ncol=dim.mat[i], rnorm(n=10*dim.mat[i]
     cpu.results[i,] <- system.time(granger.test(mat.i, p=2))}
```

Whereas this is much easier to read:

```
for(i in 1:length(dim.mat)){

  ## Define a data set with 10 rows. The for loop sets column counts from dim.mat
  mat.i <- matrix(nrow=10, ncol=dim.mat[i], rnorm(n=10*dim.mat[i], mean=0, sd=1))

  ## Evaluate the compute time for the CPU-only granger function
  cpu.results[i,] <- system.time(granger.test(mat.i, p=2))

  ## The GPU runs out of memory at larger dimensions, so comment this out for now.
  #gpu.results[i,] <- system.time(gpuGranger(mat.i, lag=2))


}
```

In addition, proper spacing gives plenty of room for comments.

These two versions indicate the other conventions to try and adhere to:

- Opening curly braces in functions and loops should always end a line. Closing curly braces should be on their own line. This makes it easier to establish where the code for the loop or function begins and ends.
- Individual lines of code should be grouped together by function. If lines of code do not logically belong together by function, they should be separated by blank lines
- Commented code should be preceded by a single #.
- Comments–that is, description of what the code file is doing–should be preceded by "## ". Notice the trailing space after the second #.
- Code inside loops or functions should be indended 2 spaces from the first letter of the loop or function definition. This provides visual separation of the code inside the function from the rest of the code file. If you use Emacs + ESS, this should be the default setting
- Line returns to separate out arguments passed to functions should be used only after commas. The difference is:

```
result.foo <- foo(x, y, z=
                  TRUE)
## vs.
result.foo <- foo(x,
                  y,
                  z=TRUE
                  )
```

5 Comments

Comment your code. You and others who read your code will benefit from clear comments that explain what is going on in different sections, clarify anything you do that is odd, and indicate how different parts of the code file relate to one another. Commenting code is part of writing code itself, not an obnoxious add-on process. People who write a lot more code than this course's instructor does suggest that writing comments should consume 25% of the entire time you spend writing the entire code file; that is, for every hour spent on a file, 15 minutes should be spent putting in comments.

All code files should start with a set of comments indicating who wrote the file, when it was written, what project it applies to, etc. An example of such a section would be:

```
## Mark Huberty
## PS239T Fall 2009
## Code for PS239T Lecture 6
## Date: 28 September 2009
```

Do not over-comment your code. Paragraphs describing what's going on are no more illuminating that the code itself, assuming you mark up that code with illuminating short descriptions of what's going on.

Other conventions to observe:

- All functions should have a comments header indicating what the function does. That header should include:
  - The purpose of the function (i.e., "Calculate the Riemann approximation of x^2")
  - The operation the function does on those inputs
  - The output the function produces

  As an example:

```
## FUNCTION riemann.xsq
## This function will take three inputs to calculate the Riemann sum for x^2:
## The number of bins, or rectangles, the area under the curve should be broken into
## The minimum value of the range over which the integral should be calculated
## The maximum value of the range over which the integral should be calculated

## The function returns the Riemann sum approxmation of the integral of x^2
```

- Do not use the first person (i.e. “Here I do `foo` to `bar`”)