CPAT Project Style Guide

Curtis Miller

August 9, 2018

Introduction

This is a style guide for the project **CPAT**, a package implementing change point analysis tests. This document defines naming conventions, whitespace conventions, etc.

Naming

Functions, Variables, Classes

- These names should be all lower-case; the exceptions are when a name is referencing some mathematical object, where the conventional reference involves a capital letter (ex. $E = mc^2$; E is conventionally capitalized, so a function related to E could use `E` rather than `e` in the name).
- So-called "camel case" and similar names separating "words" via capitalization should be avoided. Separation should be done with `_`; `.` should be avoided, but exceptions can be made when the name is trying to signal its relationship with other objects in R from other packages. For example, functions that return `htest`-class objects have names like `t.test()`, so a name like `hr.test()` that returns a `htest`-class object is appropriate.
- Names should not exceed 16 characters. Try to avoid names exceeding 8 characters, but it's more important that names be informative.
- Avoid names for existing functions or objects when possible.
- When referring to functions in comments, end the function's name with `()`; for example, `mean()`.
- C++ functions that form the "meat" of an R function (so that the R function is effectively an interface to the C++ function) should have the same name as that R function but with `_cpp` appended at the end.

Files

- File names use capitalization; "words" are separated with capitalization, not `_`, `-`, or `.`.
- A file that begins its name with "XOUT" are not meant to be included in a public distribution of the package, for whatever reason.
- R files end in `.R`; R Markdown and Seave files end in `.Rmd` and `.Rnw`, resp.
- A file that begins in "EXEC" is a file that is intended to be executable; this file should be executed by the user from the command line, *not* by any software involved with the package (ex. when loading or building in the package).
- File names should be informative.

External Packages

- External packages are referred to in documentation in boldface, like **cointReg** or **CPAT**.

R Language Conventions

- Always use `<-` for variable assignment; never use `=`.
- Curly braces (that is, `{}`) are placed in lines according to K & R style, such as

```
f <- function(x) {
  cat("x:", x)
}
```

- `;` should never be used to conclude a line. On occasion, it can be used to separate statements that are very closely related and put on the same line, such as `y <- 1 + 1; cat("y:", y)`, but this should be a rare occurance and, again, the statements should be short.
- Functions should return values by calling the value returned in the last line. `return()` should be used only when the function is terminating early.
- When defining function arguments, arguments without default values should be listed first, followed by functions with default values, and concluding with `...` if present.
- The only time `require()` can be used is when it is it is used to check if a package exists and install the package if it doesn't; for example,

```
if (!require("dplyr")) {
  install.package("dplyr")
  library(dplyr)
}
```

- In all other contexts, packages should be loaded via `library()`, and this should never be done in a `.R` file located in the `/R` directory of the package.

Lines and Whitespace

- Tab characters should never be used. Two spaces represent a tab and one level of indentation in R code. Other contexts use four spaces.
- Code blocks should be properly indented to indicate what code is in a code block. More indenting and spacing can be used to make arguments, lists, etc. visually line up, making it easier to read. For example:

```
mat <- matrix(c(1, 1, 1,
                1, 2, 3,
                1, 3, 5), nrow = 3)
```

- Function arguments in function definitions should not be defined on separate lines; they should use as few lines as possible. That said, function arguments should align in such a way that the first character of the first argument on each line aligns, causing the name of the function (and the accompanying parenthesis) has its own visual space. For example:

```
r foo <- function(bar, baz = 2, # Other arguments not written...                fin
= 10)
```

- Operators such as `<-`, `+`, `%*%`, etc. should be surrounded by spaces; for example, `y <- x + 1`, not `y<-x+1`. This includes `=` when it is being used for assigning argument values, so `f(foo = 7)` is fine, while `f(foo=7)` is not. The only exceptions are `/` and `^`; it's appropriate and common to not space these out.
- A line should not exceed 80 characters. If a line starts to become very long, consider writing a short function with most of the line's functionality.

- Code should not be separated by more than one blank line.
- Short ideas, such as an extremely simple `if` statement, can be kept to one line; otherwise, split up across lines.
- Trailing whitespace and whitespace on blank lines should be avoided and deleted when possible.
- In `DESCRIPTION`, fields listing other relevant packages/software (such as `Depends` and `Imports`) need to list each package (and relevant version information) on its own indented line.

Documentation and Comments

Documentation

- Every function should be documented using **roxygen2**-style commenting, which is placed before the function definition. See the documentation for **roxygen2** for more details. Even short functions should be documented this way. The only exceptions are anonymous functions. Functions that are defined within another function may have shorter documentation that doesn't follow **roxygen2** style.
- When documenting, spacing should be such that the description of an argument aligns with itself, like so:

```
#' @param foo A long description of the parameter, and the description will
#'            cause the line to exceed 80 characters, but the lines of the
#'            description align because of spacing
```

- Incomplete documentation should include `TODO: WRITE DESCRIPTION` or a similar `TODO` statement.
- A function should have a title, with title case capitalization. There should be a short description and an optional long description. Descriptions of the function can have full sentences ending with periods; parameter and return value descriptions will not be sentences and will not be concluded by or contain periods. There must be an example of function usage.
- Every `.R` file needs to start with a description. The first line can be a shebang for executable files. But this should be immediately followed by the following:

```
################################################################################
# MyFile.R
################################################################################
# 2018-12-31 (last modified date)
# John Doe (author)
################################################################################
# This is a one-line description of the file.
################################################################################

x <- 1 + 1  # First line of code
```

Comments

- Not every line needs to be commented; do so when it's not entirely clear what a line does; such as implementing some statistical procedure. That said, aim for self-documenting code rather than lots of comments.
- Major sections should be separated with 80 `#`, followed by `# TITLE` on a new line, then another 80 `#`.

- A comment can share a line with code but the line should not exceed 80 characters even with the comment. It needs to be at least two characters away from the last language-relevant character. More spaces can be used if this would cause comments to align in a way that improves readability.
- Multi-line comments should precede the relevant code; they should not share lines with code.
- It is appropriate to comment out unused or deprecated code and it does not need to be deleted even in deployed code.
- `TODO` comments are to be inserted to show what further work needs to be done in a document. For R scripts, they take the following format:

```
# TODO: user: WHAT NEEDS TO BE DONE -- Mon 31 Dec 2099 11:59:59 PM GMT
```