CSC 121: Computer Science for Statistics

Radford M. Neal, University of Toronto, 2017

http://www.cs.utoronto.ca/∼radford/csc121/

Week 13

A Few Final Comments

Here in the last lecture, I'll mention a few things that we haven't had time to really cover...

- More on R packages.
- More on testing your program.
- Managing versions of your program, and writing a program together with other people.
- Programming languages other than R.

R Packages

The basic features of R can be extended using "packages" of function definitions, data sets, etc.

These packages are available in several ways:

- A few packages come with R and are available for use by default. One such is the `stats` package that defines the `lm` function.
- Some more packages come with R, but have to be loaded manually before you can use them. An example is the `survival` package for analysing survival data. To use it, say `library(survival)`.
- Many other packages (thousands) are available for installation from package repositories, to which many people have contributed. The CRAN repository is the best-known, and is the default when you try to install such packages using the `install.packages` function.
- You can write your own packages.

Some Things Packages May Do

- Provide general extensions to the R language.
  Example: `magrittr` defines an operator `%>%` for "piping" data from one function to another.
- Provide convenient ways of interacting with R.
  Example: `knitr`, which we've been using to produce convenient output.
- Interface to other software.
  Example: `foreign` provides ways of reading data from other statistics packages (eg, SAS) into R.
- Provide more elaborate ways of producing graphical output.
  Example: `ggplot2` is a very popular way of producing graphs.
- Provide additional statistical methods.
  Example: `tgp` implements "Treed Gaussian Process" models, that can model how a response variable relates to explanatory variables more flexibly than a linear model.

Program Testing

In the assignments, you've been creating some tests for the functions you write. Creating a good set of tests is important when first writing a program, and also when making changes to the program later, to check that you haven't broken it.

There are two general kinds of tests one might do:

- Tests that the whole program works as intended.

  This is what we really care about. But it may be hard to think of ways to test everything about a program at this level. And it's particularly hard to test programs that produce plots for a user to view, or that interact with a user.

- Tests that individual functions work as intended, including functions that are just used inside the program (aren't meant to be used elsewhere).

  If we are confident that many of the parts of the program work correctly, we will be more confident that the program as a whole works correctly.

One aim in testing is to make sure that every bit of code has been used — eg, that every `if` statement has been tried with the condition being both `TRUE` and `FALSE`. But that's not enough to guarantee that the program always works. Testing isn't a substitute for careful design and coding.

Source Code Control

A *source code control system* manages the files containing your function definitions, scripts, or documentation.

Here are some things a source code control system lets you do:

- Go back to an earlier version if you find out that some recent changes you made were a bad idea.
- See what has changed from some earlier version to the current version.
- Create multiple versions of a program, perhaps specialized for slightly different tasks.
- Merge work on one project that is done by several programmers.

Currently, the most popular source code control system is `git`. It is supported by RStudio, or it can be used on its own.

Source Code Repositories

It's increasingly popular for programs (managed by a source code control system) to be made available to everyone on *source code repositories*.

Two popular ones based on `git` are `gitlab.com` and `github.com`.

These repositories support

- The developers uploading programs, including changes to previous versions.
- People downloading the programs, including the revision history if they wish.
- People reporting bugs.
- People submitting changes to programs to be considered by the developers.

Of course, the developers have to provide a license that allows the program to be used / changed.

Other Programming Languages for Statistics

R is probably the most common programming language used by statisticians. But there are others.

There are statistical packages that provide programming facilities, such as

- SAS
- Stata

There are several programming languages with wider communities that are somewhat similar to R, including

- Matlab (and its free version, Octave).
- Python

There are also languages centred on symbolic mathematical computation, like

- Maxima
- Maple
- Mathematica

In these languages, you can multiply `2+x` by `1+3*x` and get `2+7*x+3*x^2`.

Compiled Programming Languages

There are also programming languages that are usually *compiled*, rather than *interpreted*, like R, and the other languages on the previous slide.

Compilation translates the program to a program in *machine language*, which the computer can do directly. In contrast, an interpreter is a program in machine language (usually compiled from some other language) that looks at a program and does what it says, which is much slower.

So if you need your program to go really fast, you may want to write it in a language that can be compiled, rather than in R.

Some common compiled languages:

- C
- C++ (like C but with object-oriented programming facilities)
- Fortran

You can also write just the time-critical part of the program in one of these languages, and then call that part from an R program.

Alternative Implementations of R

Several projects are currently in the works to improve on the current implementation of R (as distributed at `http://R-project.org`).

These include:

- FastR, supported by Oracle: `https://github.com/graalvm/FastR`
- Rho, supported by Google: `https://github.com/rho-devel/Rho`
- pqR, my own effort: `http://pqR-project.org`

  My original aim was to create a faster version of the R interpreter.

  I have also begun to extend the R language in ways that make it more useful, and fix some of its design flaws.