Why you should teach your stats students C

Ben Klemens

5 September 2011

[This is an essay for those of you who are teaching a nontrivial amount of programming to your students. If you're teaching just enough programming to run the `regress()` function and absolutely no more, then don't worry about all this.]

I'll start with the main argument as to why you shouldn't teach your stats students C. Here are some snippets cut and pasted from Joel[1], a guru well known in the straight-up programming world.

> . . . there are two things traditionally taught in universities as a part of a computer science curriculum which many people just never really fully comprehend: pointers and recursion.
>
> All the kids who did great in high school writing pong games in BASIC for their Apple II would get to college, take CompSci 101, a data structures course, and when they hit the pointers business their brains would just totally explode, and the next thing you knew, they were majoring in Political Science because law school seemed like a better idea. I've seen all kinds of figures for drop-out rates in CS and they're usually between 40% and 70%.
>
> When you struggle with [more quotidian programming issues], your program still works, it's just sort of hard to maintain. Allegedly. But when you struggle with pointers, your program produces the line **Segmentation Fault** and you have no idea what's going on. . .

For applied statisticians, the conversation typically ends there: C is hard because it doesn't hide pointers, and there are other languages where you don't have to think about them.[2]

The other objection, which I'll put in as an aside, is that the environment for C is entirely open, while closed environments are easier to get started with. R, matlab, &c. provide you with a single window where the commands and the output go. I think this is an outdated objection, and if you poke around, you'll find that installing a full development environment is about as easy as installing a stats package, and that IDEs and stats package GUIs have basically merged in functionality. There are so many tools

[1] `http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html`

[2] I sometimes wonder if it isn't just that *pointers pointing to data* sounds so sharp, and maybe they need another name. "An array is implemented as a bunny paw cuddling onto the first element of the array."

for tracking down those segmentation faults in the present day that they're not really such an issue anymore.

Back to that main thread about pointers, which are the location of data rather than data itself. Why is there any value to making that distinction, and working in a language where users have to think about it? From a practical perspective, there's the simple fact that pointers will speed up your work immensely, so you can bootstrap variances from your MCMC model without tears.

But from a theoretical perspective, statistics is all about multiple levels of references to data. Above, the CompSci students had trouble with understanding the difference between data and the location of data, but if you've ever taught Stats 101, you've spent a class period watching your students get mystified by how the variance of the data and the variance of the mean of the data are different. Then you get to do that over when you teach regressions and show the students that the variance of the data, the variance of the OLS parameters, and the variance of the error term are again all different things.

That is, statistics is filled with distinctions between data, statistics of data, and statistics of statistics of data. Let's say we have a simple hierarchical model, where we take the mean of each subgroup, then run a Probit on the means. The variance of those Probit parameters are statistics of statistics of statistics of data. You'll sometimes find box-and-arrow diagrams of such models that look a lot like the diagrams used to teach pointers.

So when your students are learning C and getting lost about a pointer to a pointer to data, they are getting practice in exactly the same skill they need to keep track of what's going on in nontrivial statistical models.

If you clicked through to the article by Joel above, then you saw that Joel is actually pretty pro-C as a teaching language:

> But when you struggle with pointers, your program produces the line **Segmentation Fault** and you have no idea what's going on, until you stop and take a deep breath and really try to force your mind to work at two different levels of abstraction simultaneously.

Earlier in the essay, Joel explains that "You need training to think of things at multiple levels of abstraction simultaneously, and that kind of thinking is exactly what you need to design great software architecture." It's also exactly what you need to understand a hierarchical model or even the difference between the variance of a mean and the variance of a data set. If your students are smart enough to understand statistics, they're smart enough to understand pointers. And after they get good with pointers, learning statistics will be easier.

And, as a bonus, the code your students write will be of higher quality, because they'll be writing with a full tool set, not the subset that a stats package offers under the presumption that users aren't smart enough to work with both data and references to data.

The other reason I like C as a teaching language is that it is a very simple language, with few grammatical exceptions for anything. I do all my C work with 18 keywords (which is a count way at the bottom of the rankings, which typically range between

about forty and 'we stopped counting'). I'll get to the scoping constructs next time, but they're also darn simple. You won't spend another four or five sessions of class time going over objects, encapsulation, and exceptions to the object and encapsulation hierarchy, because these things are done without additional syntax. The complexity of real-world problems—string handling, matrices, vectors, regressions—happens via libraries that load more functions and structures, but leave the basic syntax intact.

In fact, the great majority of other languages and packages could be described with a sentence of the form, *this language is kinda like C, except it has an additional syntax to handle [elements] more easily.* Which is why C is a great teaching language for students who are likely to face a half-dozen other little programming languages by the time they leave grad school: a student who learns C will have the background needed to pick up all the other languages quickly and easily. So not only will they have more training in the sort of pointer-like multiple indirection that is a modern statistical model, they'll be ready to implement it in whatever tools are needed for today's project.