Calling C/C++ Code in

Created By Matt Branan on Wednesday, September 17, 2014 1:13:58 AM MDT
last modified by Matt Branan on Wednesday, September 17, 2014 1:28:34 AM MDT

As R programmers, we rely on our high-level language for most of our interactions with statistical coding. However, sometimes we would like to use lower-level languages for tasks such as speeding up code. R programmers are told to vectorize our functions whenever they can while working within R, but if there is no way around those dreaded for/while loops (e.g., autoregressive models, MCMC routines) then one route to take would be to push that bit of code to the lower-level C or FORTRAN family of languages. John Nash, the author of the optim() and optimx() functions has said that he typically pushes his objective functions that he is optimizing to C because these functions are evaluated a ton of times and it can be quite computationally expensive.

For this first bit, I will sweep the problem of actually knowing some C/C++/FORTRAN code under the rug and assume that we all know how to code in these languages. Also, when you see "C", you can probably substitute "C++" into the statement as well without losing anything.

There are several functions that one can use within R to call outside functions written in other languages:

- .C() -- This is the "older" function to call C functions and requires the user to manually handle a lot of the underworkings of C code and can be less efficient (e.g., matching C object types to R object types is a big issue here)
- .Call() -- This is the "newer" function to call C functions without as much manual tweaking of object types and does the translation more efficiently
- .Fortran() -- A function to run FORTRAN code directly in R
- .External() -- Similar to .Call(), this function is better able to handle argument lists that are not of the exact length required by the called function (e.g., when you have nested functions in R, we use the ",..." argument to allow varying argument list lengths and .External() is similar in this respect)

I'll focus on using .Call(), but check out your favorite functions in R by typing just their name at the prompt and search through the R source code to find where the other four functions are used (optimize() and filter() being two of them).

First, you will want to write up a C program in a text editor and saved it as *myprogram.c*. Then, you'll want to compile it. On the OSX operating system, I did this by opening up the Terminal, going to the directory in which my C program was stored, and typed in the following at the prompt.

R CMD SHLIB *myprogram.c*

This compiles the C program and creates *myprogram.o* and*myprogram.so* files in your directory. Then, you can open up R, use setwd() to change your working directory to the one in which your C program is saved, and use the function

dyn.load("*myprogram.so*")

Ah ha! Now, we can get to the actual calling of your carefully-crafted function with

.Call("*myprogram*",...)

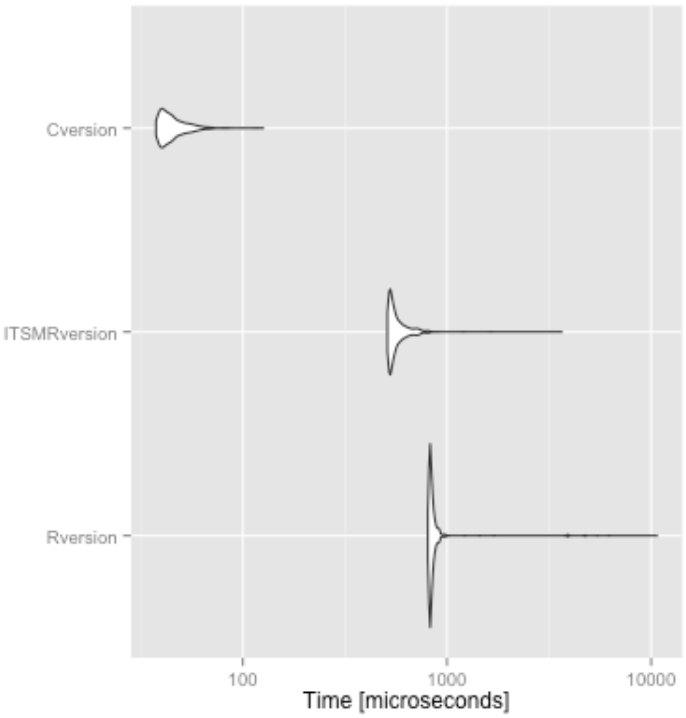
Where the ",..." stands for the arguments you need to pass to your C program. Viola! You've successfully called a C function from within R!

Below, I offer an example of one such program I've written. The C program below simply generates an autoregressive process: $X_t = 0.7 * X_{t-1} - 0.1 * X_{t-2} + Z_t \; ; \; for \; Z \sim N(0,1)$.

1. First, I write up the program and name it argenerator.c.
2. Then, I open up the Terminal and run the command: R CMD SHLIB argenerator.c (making sure I'm in the right directory!) and this makes argenerator.o and argenerator.so.
3. Opening up R, I then type the commands > dyn.load("argenerator.so") to get the code into R's memory and then >.Call("argenerator",c(0.7,-0.1),rnorm(2,0,1),rnorm(100,0,1)) -- because the three arguments to argenerator are the parameter vector, the first two values of the process, and the white noise term, in that order.

4. This returns my autoregressive process and I can check it with a function that I wrote in R to make sure that I've done everything correctly (this checking step is highly recommended!).

What kinds of improvements over the plain-ol' R functions can we expect from all of this extra C work? I've coded up one version of generating the autoregressive process in R only and one in C and called it inside of R. Then, I used the package microbenchmark to compare the time it takes to run each of the functions based on 1,000 calls to each of the functions. I've also included the sim() function from the istmr package that does the same thing. The plot of the results follows below and the R code can be found here. The main point is that the Cversion of the function was noticably quicker to execute than either of the other two versions (about 19 times faster than the Rversion and about 12 times faster than the ITSMRversion based on just the median times it took to run each function).



Some quick and dirty tips when using C code:

- Vector elements are numbered starting with "0"! In R, we start counting vector elements at "1".
- You must be careful to give every object in your C program an object type (read up in the links below on the SEXP object type).
- End every line with code with a semi-colon (;)
- When compiling the C code in OSX, make sure that you have the Developer Tools (XCode, for example)

Finally, here are some references for those who would like to learn more about coding in other languages, in general, and about running outside languages in R, specifically.

- Code Academy: for learning just about any programming language
- Learn X in Y Minutes: another site for introductions to many programming languages
- Writing R Extensions: general instructions on importing outside code into R from the R Development Team (dense, but has just about everything that you could want to know about it)
- Using .Call in R: a nice, light, introduction to using the .Call() function from Brian Caffo
- Language Interfaces - .Call and .External: another light introduction to .Call() and .External from Peter Dalgaard
- Using R - .Call("hello"): Jonathan Callahan compares .C() and .Call() and has a few great links in there as well