An introduction to *Rcpp*

R and **C++** integration for performant algorithms

Sean Wu, Epidemiology PhD Student
https://slwu89.github.io

Schedule

Link to GitHub repo: https://github.com/slwu89/rcpp_ccb_seminar

1. Brief discussion of *R*, *C++*, and *Rcpp*
2. An example of how to translate a small algorithm to *C++*
    a. Introduction to the example
    b. 2 translations, one more simple, the other for in-depth
3. How to run *Rcpp* functions in parallel *R* through *foreach* parallel interface
4. Pitfalls when making packages that include *C++*
5. Introduce *RcppArmadillo* & *RcppGSL*
6. Other curiosities that are good to know the existence of
7. Resources

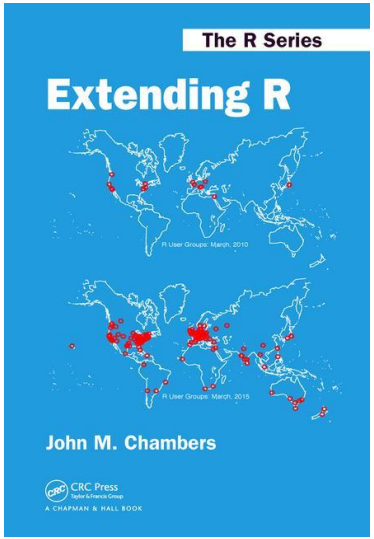
Conspicuous absences

- Debugging
- Intro to *C++*
- Things that other people have explained better than me on the internet (Hadley Wickham...)

R and *C++*

- Easy to write yet slow
- *R* doesn't care what you do
- Has garbage collection

- Hard to write yet fast
- *C++* cares about *everything* you do
- Has pointers
  - Although see smart pointers in *C++11/14*



The role of *R* as an interface language between algorithms and users/GUIs is explored more fully in this book, if you're interested.

Rcpp

What you need to know to get started right away:

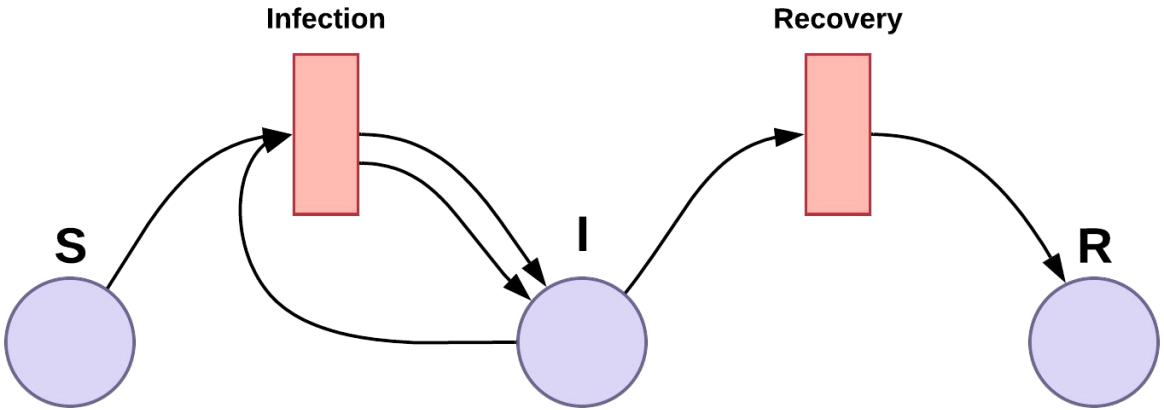
- Relationship between the core *R* data types and *Rcpp* data types
- How to convert data between *Rcpp/R* and *C++* types (both builtin and *some* STL containers)
  - `Rcpp::as<C++ thing>(R thing)` : gets you from *R* to a *C++* type (useful for parsing R arguments to C++ functions)
  - `Rcpp::wrap(C++ thing)` : gets you from *C++* to *R* type (useful when returning from functions)
    - This is usually called implicitly, but there is an example in the package included with the repo
- How to get random numbers (we will cover this)
- How to load your *C++* code into something *R* can call (we will cover this)

For small projects, *RStudio* is a perfectly acceptable *C++* IDE (it will link to clang or gcc to give static code analysis).

Example: writing a Gillespie algorithm in C++

… but first, some epidemiology (sort of)

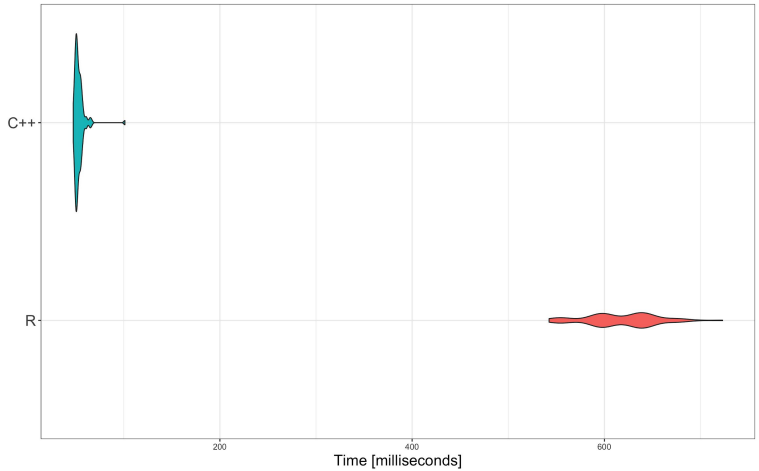
- SIR model is a basic model of simple disease transmission (the *Ising model* for epidemiologists, if you will…)
- Can be mathematically represented as a CTMC
- A handy modelling formalism for simulating trajectories from many CTMCs on a computer is a Stochastic Petri Net (SPN)
    - Gillespie's algorithm is a method to sample trajectories from this model



Example: writing a Gillespie algorithm in C++

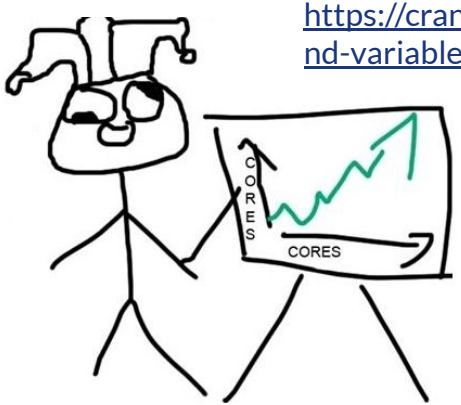
This is a situation where including some *C++* can significantly improve performance.

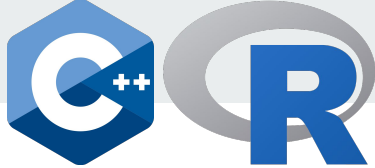
- Basic algorithm design is close to common statistical algorithms (MCMC, EM, iterative , etc) --- a while loop with a termination condition.
- We will examine a *R* implementation, a quick *Rcpp/C++* implementation, and a more involved *C++* implementation that would not look out of place in a standard *C++* projects source code.
    - The repo also has a package which wraps the code, that we will look at later.



Running C++ code in parallel via *foreach*

- Why are we talking about this?
  - `task 1 failed - "NULL value passed as symbol address"`
- Why is this happening?!
  - If the C++ function was only compiled in R's global environment, the node in the cluster won't know about it!
    - By default, they get exported the "symbol" of the function but not the shared object that was compiled, or that's my understanding.
    - See this for an obscure "explanation": https://cran.r-project.org/doc/manuals/r-devel/R-ints.html#Environments-and-variable-lookup





Building packages with *Rcpp*

- Common pitfalls (can't find the dynamic library):
  - `Error in .Call("myfunction", PACKAGE = "mypackage", : "myfunction" not available for .Call() for package "mypackage"`

- Checklist for compiling successfully:
  - R/mypackage.R: tell it to use the dynamic library of your package
  - src/Makevars & src/Makevars.win: if you have things you need to tell the C++ compiler (eg; where to find header files)
  - DESCRIPTION: needs to have "Imports: Rcpp" and "LinkingTo: Rcpp" so the package build system can find everything it needs
  - If you are using roxygen2 to build documentation, remember that in order to export functions, you need to have *both* these tags before the C++ function:

    ```
    //' @export
    // [[Rcpp::export]]
    ```

    - The first builds the roxygen docs and R's `.Call` wrapper, the second registers the exported functions so `.Call` can find it.

RcppArmadillo & RcppGSL

These are nice software libraries that very smart people have spent a lot of time thinking hard about.

RcppArmadillo

- Pleasant to use and reasonably fast library for matrix algebra (sparse & dense)
  - Supports 3-tensors (cubes)
- Some embarrassingly bad/old code using *Armadillo* for adaptive MCMC (for matrix algebra for updating covariance matrix of proposal matrix)
  https://github.com/slwu89/MCMC/blob/master/adaptMCMC_source.cpp

RcppGSL

- A bestiary of strange things useful for applied maths and statistics
  - ODEs, optimization, random numbers, combinatorics, numerical maths
  - C-style API, somewhat user unfriendly (remember to free memory when you're done with it!)
  - Blazing fast, nerd street cred

Other things that exist

- *RcppEigen*
  - Less user-friendly API than *RcppArmadillo*, less stackoverflow banter about it too
  - Claims to be faster
- *BH (Boost Headers)*
  - Provides large chunks of the Boost project's header-only libraries for *Rcpp* powered projects
  - As far as I can tell, Boost is currently functioning as C++'s weird backyard where things get tested before incorporation into the ISO C++ standard
- *RcppR6*
  - For the OOP obsessed, you can directly export C++ classes to R6 classes: https://github.com/richfitz/RcppR6
- *OpenMP*
  - If C++ *still isn't fast enough* you can try out OpenMP via a *Rcpp* plugin
  - I tested this before on High Sierra: https://github.com/slwu89/RcppOMP
    - If this code melts your processor it's not my fault
- Debugging
  - The bane of a C++ programmer's existence. See http://kevinushey.github.io/blog/2015/04/13/debugging-with-lldb/ for hints on *Rcpp*-centric debugging.

Resources

- *Rcpp* on Mac
  - The Mac environment generates many inscrutable problems when using *Rcpp*. See http://www.thecoatlessprofessor.com/programming/r-compiler-tools-for-rcpp-on-macos/ for advice.
- *Rcpp* on Windows
  - You will need to use Rtools https://cran.r-project.org/bin/windows/Rtools/
- Improving *C++* knowledge
  - Anything by Scott Meyers: https://www.aristeia.com/books.html (the "Effective C++" series is excellent)
  - "Modern C++ Design: Generic Programming and Design Patterns Applied" by Andrei Alexandrescu
    - Kind of esoteric, some parts aren't ageing well but if questions regarding "did I choose the right *design pattern*?" keep you up at night this may help
  - "Programming Game AI by Example" by Matt Buckland
    - Actually a great walk through of how to build a relatively complex game system step by step in C++
- Improving *R* knowledge
  - "Advanced R" of course http://adv-r.had.co.nz
  - Anything by John Chambers
  - "R Programming for Bioinformatics" by Robert Gentleman
    - Chapter 6: Foreign Language Interfaces describes how R links to C, which can help demystify some of the stranger compiler errors *Rcpp* may spit out from time to time.