M+E+C:
Computation with **R**

Keith O'Hara

01/18/2018

R in a Nutshell

- **R** is a modern, open-source implementation of the **S** programming language.
- Written mostly in `C`
- **R** is designed to be a mixture of interactive and OO-style programming
  - 'Extreme dynamism'
  - Classes and methods
- It's a mature project... which is both a good and bad thing
  - *e.g.*, not designed with parallel programming in mind
  - started in 1993 by **R**obert Gentleman and **R**oss Ihaka
  - version 1.0.0 released in 2000
- R-Core
- RStudio

R features

OOP and **R**

- Not quite OOP
- Usual case:

  object.method("do something")

  For example,

  model.solve(parameters)

  That is, the solve method depends on the model type.
- **R** (S3):

  method(object, "do something")

OOP and **R**

- OO systems in **R**:
  - S3 (informal; ad hoc; most commonly used)
  - S4 (more formal; multiple dispatch; awkward)
  - Reference Class (conforms more to usual message-passing OO systems); looks like object$method("do something")
- S4:

  object@member

- Not as widely used as S3. Bioconductor is a well-known collection of S4 packages.

Peculiarities of **R**

- "**R** is slow..." limited BLAS and LAPACK
  - Build using OpenBLAS or system libraries (such as vecLib)
- The number of function inputs affects performance (even if they're not used); *e.g.*,

  inner_product(x,y,blah,blah,blah,blah,blah,blah)

  This is generally not a feature of compiled programming languages.

  This is due to 'lazy evaluation.' Example:

```
foo <- function(x, y=z) {z <- x*x; y*log(z)/x}
foo(2); foo(2,1)
```

- Vectors and matrices (v.s. Matlab)
- C API
- $<-$ vs $=$

R Packages

- Great package system!

  The Comprehensive R Archive Network (CRAN): 12069 packages
- Bioconductor
- Some useful packages:
  - devtools
  - Rcpp
  - RcppArmadillo
  - ggplot2

R Packages: Building from Source

- Building **R** packages from source requires some tools.
  - Windows users should install Rtools.
  - Mac users should install Xcode (and get gfortran).
- **R** is built using gfortran v4.8.2 or v6.
- Example:

```
install.packages("devtools")
library(devtools)
install_github("TraME-Project/Shortest-Path-R")
```

- **Let's look at the structure of a package!**

R Internals

S-Expressions (SEXP)

- All R objects are declared as 'SEXP' objects when passed as inputs on a `C`-level.
- You can pass pretty much anything as a 'SEXP' object; you can even use it to call R functions from `C/C++` code.

```
SEXP add_one (SEXP a_R, SEXP func)
{
    try {
        Function myFunc = as<Function>(func);
        NumericVector a = as<NumericVector>(a_R);
        NumericVector b = myFunc(a);
        //
        return wrap(b);
    } catch( std::exception &ex ) {
        forward_exception_to_r( ex );
    } catch(...) {
        ::Rf_error( "C++ exception (unknown reason)" );
    }
    return R_NilValue;
}
```

Rcpp and RcppArmadillo

- Rcpp is a great package with an easy to use (and abuse) API for working with C++ and R.
  - Basic objects: Rcpp::NumericVector b and Rcpp::List::create(Rcpp::Named("b") = b);
- RcppArmadillo is essentially a skeleton package that contains the Armadillo header files.
- Use Rcpp; avoid using R's `C` API directly.
- Load the Rcpp package and call dynamic loaded code using

```
dyn.load("simp_test.so")

.Call("my_C_function",input_1,input_2)
```

Parallel Computing

- 'parallel' package; combines snow and others
- How it works; memory issues
- 'foreach' function
- Passing current environment and other functions
- Tip for working with R on NYU's HPC (nodes vs cores)

Parallel Computing: Example

```
library(doParallel)
#
n_cores <- 2
#
cl <- makeCluster(n_cores)
registerDoParallel(cl)

kk <- foreach(i=1:8, .combine=c) %dopar% rnorm(i*10)

stopCluster(cl)
```

- Options: `.inorder`, `.packages`, `.export`, `.noexport`