COMPUTATIONAL STATISTICS IN PRACTICE

SOME OBSERVATIONS

Dirk Eddelbuettel

10 November 2016

Invited Presentation
Department of Statistics
University of Illinois
Urbana-Champaign, IL
Motivation
Almost all topics in twenty-first-century statistics are now computer-dependent [...] 

Here and in all our examples we are employing the language R, itself one of the key developments in computer-based statistical methodology.

Efron and Hastie, 2016, pages xv and 6 (footnote 3)
Computational Statistics in Practice

- Statistics is now computational (Efron & Hastie, 2016)
- Within (computational) statistics, reigning tool is R
- Given R, Rcpp key for two angles:
  - *Performance* always matters, ease of use a sweetspot
  - “Extending R” (Chambers, 2016)
- Time permitting
  - *Being nice to other (languages)*
  - an underdiscussed angle in industry
Drawing on three Talks

• Rcpp Introduction (from recent workshops / talks / courses)
• [if time] If You Can’t Beat ’em (from JSS session at JSM)
• [if time] Open Source Finance (from an industry conference)
Brief Bio

- PhD, MA Econometrics; MSc Ind.Eng. (Comp.Sci./OR)
- Finance Quant for 20 years
- Open Source for 22 years
  - Debian developer
  - R package author / contributor
- R Foundation Board member, R Consortium ISC member
- JSS Associate Editor
Rcpp: Introduction via Tweets
Using #Rcpp to leverage the speed of C++ with the ease and clarity of R. Thanks, @eddelbuettel
Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois #rstats
Thanks to @eddelbuettel's Rcpp and @hadleywickham AdvancedR Rcpp chapter I just sped things up 750x. You both rock.
Writing some code using #rstats plain C API and realising/remembering quite how much work Rcpp saves - thanks @eddelbuettel
"Rcpp is one of the 3 things that changed how I write rstats code". @hadleywickham at #EARL2014
@eddelbuettel @romain_francois Have I emphasized how much I ❤️ #Rcpp?
Gosh, Rcpp is the bee's knees (cc: @eddelbuettel) #rstats
The rise of Rcpp #rstats
"It's easier to make an error if I am not using Rcpp"
-- @GaborCsardi, right now in the (wicked) R Hub presentation
EXTENDING R
Why R? : Programming with Data from 1977 to 2016


UIUC Nov 2016
xx <- faithful[, "eruptions"]
fit <- density(xx)
plot(fit)
A Simple Example

\begin{equation}
density.default(x = xx)
\end{equation}

Density

N = 272   Bandwidth = 0.3348

UIUC Nov 2016
xx <- faithful[, "eruptions"]
fit1 <- density(xx)
fit2 <- replicate(10000, {
  x <- sample(xx, replace=TRUE);
  density(x, from=min(fit1$x), to=max(fit1$x))$y
})
fit3 <- apply(fit2, 1, quantile, c(0.025, 0.975))
plot(fit1, ylim=range(fit3))
polygon(c(fit1$x, rev(fit1$x)), c(fit3[1,], rev(fit3[2,])), col='grey', border=F)
lines(fit1)
density.default(x = xx)

N = 272   Bandwidth = 0.3348
R enables us to

- work interactively
- explore and visualize data
- access, retrieve and/or generate data
- summarize and report into pdf, html, ...

making it the key language for statistical computing, and a preferred environment for many data analysts.
R has always been extensible via

- C via a bare-bones interface described in *Writing R Extensions*
- Fortran which is also used internally by R
- Java via *rJava* by Simon Urbanek
- C++ but essentially at the bare-bones level of C

So while *in theory* this always worked – it was tedious *in practice*
Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

*Since the core of R is in fact a program written in the C language, it’s not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*
Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

*Since the core of R is in fact a program written in the C language, it’s not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*
Chambers proceeds with this rough map of the road ahead:

- **Against:**
  - It’s more work
  - Bugs will bite
  - Potential platform dependency
  - Less readable software

- **In Favor:**
  - New and trusted computations
  - Speed
  - Object references
The *Why?* boils down to:

- **speed**: Often a good enough reason for us ... and a focus for us in this workshop.
- **new things**: We can bind to libraries and tools that would otherwise be unavailable in R
- **references**: Chambers quote from 2008 foreshadowed the work on *Reference Classes* now in R and built upon via Rcpp Modules, Rcpp Classes (and also RcppR6)
And Why C++?

• Asking Google leads to tens of million of hits.
• Wikipedia: C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language
• C++ is industrial-strength, vendor-independent, widely-used, and still evolving
• In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API
• As a widely used language it also has good tool support (debuggers, profilers, code analysis)
Scott Meyers: *View C++ as a federation of languages*

- C provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.
- *Object-Oriented C++* (maybe just to provide endless discussions about exactly what OO is or should be)
- *Templated C++* which is mighty powerful; template meta programming unequalled in other languages.
- *The Standard Template Library (STL)* is a specific template library which is powerful but has its own conventions.
- *C++11* and *C++14* (and beyond) add enough to be called a fifth language.

NB: Meyers original list of four languages appeared years before C++11.
Why C++?

- Mature yet current
- Strong performance focus:
  - You don’t pay for what you don’t use
  - Leave no room for another language between the machine level and C++
- Yet also powerfully abstract and high-level
- C++11 + C++14 are a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded
INTERFACE VISION
Bell Labs, May 1976

ABC Interface

ABC: general algorithm

XABC: FORTRAN subroutine to provide interface between ABC and/or language and/or utility programs

XABC (INSTR, OUTSTR)

Input INSTR →

Output OUTSTR →

Note: Names are meaningful to algorithm, not necessarily to language.
R offers us the best of both worlds:

- **Compiled** code with
  - Access to proven libraries and algorithms in C/C++/Fortran
  - Extremely high performance (in both serial and parallel modes)
- **Interpreted** code with
  - A high-level language made for *Programming with Data*
  - An interactive workflow for data analysis
  - Support for rapid prototyping, research, and experimentation
Why Rcpp?

- **Easy to learn** as it really does not have to be that complicated – we will see numerous few examples
- **Easy to use** as it avoids build and OS system complexities thanks to the R infrastructure
- **Expressive** as it allows for *vectorised* C++ using *Rcpp Sugar*
- **Seamless** access to all R objects: vector, matrix, list, S3/S4/RefClass, Environment, Function, ...
- **Speed gains** for a variety of tasks Rcpp excels precisely where R struggles: loops, function calls, ...
- **Extensions** greatly facilitates access to external libraries using eg *Rcpp modules*
Speed
Consider a function defined as

\[ f(n) \text{ such that } \begin{cases} 
  n & \text{when } n < 2 \\
  f(n-1) + f(n-2) & \text{when } n \geq 2 
\end{cases} \]
R implementation and use:

```r
f <- function(n) {
  if (n < 2) return(n)
  return(f(n-1) + f(n-2))
}
```

```r
## Using it on first 11 arguments
sapply(0:10, f)
```

```r
## [1] 0 1 1 2 3 5 8 13 21 34 55
```
Timing:

```r
library(rbenchmark)
benchmark(f(10), f(15), f(20))[, 1:4]
```

<table>
<thead>
<tr>
<th></th>
<th>test</th>
<th>replications</th>
<th>elapsed</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f(10)</td>
<td>100</td>
<td>0.016</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>f(15)</td>
<td>100</td>
<td>0.140</td>
<td>8.750</td>
</tr>
<tr>
<td>3</td>
<td>f(20)</td>
<td>100</td>
<td>1.505</td>
<td>94.063</td>
</tr>
</tbody>
</table>
A C or C++ solution can be equally simple

```c
int g(int n) {
    if (n < 2) return (n);
    return (g(n-1) + g(n-2));
}
```

But how do we call it from R?
But Rcpp makes this *much* easier:

```cpp
Rcpp::cppFunction("int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2)); }")
sapply(0:10, g)
```

```r
## [1]  0  1  1  2  3  5  8 13 21 34 55
```
Timing:

```r
Rcpp::cppFunction("int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2)); }")
library(rbenchmark)
benchmark(f(25), g(25), order="relative")[,1:4]
```

```
## test replications elapsed relative
## 2  g(25)    100  0.035 1.0
## 1  f(25)    100 16.037 458.2
```

A nice gain of a few orders of magnitude.
Run-time performance is just one example.

*Time to code* is another metric.

We feel quite strongly that helps you code more succinctly, leading to fewer bugs and faster development.

A good environment helps. RStudio integrates R and C++ development quite nicely (eg the compiler error message parsing is very helpful) and also helps with package building.
library(pagerank)  # github.com/andrie/pagerank

cran <- "http://cloud.r-project.org"

pr <- compute_pagerank(cran)
round(100*pr[1:5], 3)

##
## Rcpp   MASS  ggplot2  Matrix  mvtnorm
## 2.541   1.719   1.105   0.905   0.726
Rcpp: A Better C API for R
In a nutshell:

- R is a C program, and C programs can be extended
- R exposes an API with C functions and MACROS
- R also supports C++ out of the box with `.cpp` extension
- R provides several calling conventions:
  - `.C()` provides the first interface, is fairly limited, and discouraged
  - `.Call()` provides access to R objects at the C level
  - `.External()` and `.Fortran()` exist but can be ignored
- We will use `.Call()` exclusively
At the C level, everything is a SEXP, and every .Call() access uses this interface pattern:

```c
SEXP foo(SEXP x1, SEXP x2){
  ...
}
```

which can be called from R via

```r
.Call("foo", var1, var2)
```

Note that we need to compile, and link, and load, this manually in wasy which are OS-dependent.
#include <R.h>
#include <Rinternals.h>

SEXP convolve2(SEXP a, SEXP b) {
    int na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    a = PROTECT(coerceVector(a, REALSXP));
    b = PROTECT(coerceVector(b, REALSXP));
    na = length(a);
    nb = length(b);
    nab = na + nb - 1;
    ab = PROTECT(allocVector(REALSXP, nab));
    xa = REAL(a);
    xb = REAL(b);
    xab = REAL(ab);
    for (int i = 0; i < nab; i++)
        xab[i] = 0.0;
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return ab;
}
EXAMPLE: CONVOLUTION

```cpp
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector
convolve2cpp(Rcpp::NumericVector a,
             Rcpp::NumericVector b) {
    int na = a.length(), nb = b.length();
    Rcpp::NumericVector ab(na + nb - 1);
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            ab[i + j] += a[i] * b[j];
    return(ab);
}
```
Types Overview: RObject

- The **RObject** can be thought of as a basic class behind many of the key classes in the Rcpp API.
- **RObject** (and our core classes) provide a thin wrapper around SEXP objects
- This is sometimes called a *proxy object* as we do not copy the R object.
- **RObject** manages the life cycle, the object is protected from garbage collection while in scope—so we do not have to do memory management.
- Core classes define several member common functions common to all objects (e.g. `isS4()`, `attributeNames`, ...); classes then add their specific member functions.
### Overview of Classes: Comparison

<table>
<thead>
<tr>
<th>Rcpp class</th>
<th>R typeof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Numeric(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Logical(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Character(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Raw(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Complex(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>List</td>
<td>list (aka generic vectors) ...</td>
</tr>
<tr>
<td>Expression(Vector</td>
<td>Matrix)</td>
</tr>
<tr>
<td>Environment</td>
<td>environment</td>
</tr>
<tr>
<td>Function</td>
<td>function</td>
</tr>
<tr>
<td>XPtr</td>
<td>externalptr</td>
</tr>
<tr>
<td>Language</td>
<td>language</td>
</tr>
<tr>
<td>S4</td>
<td>S4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
• `IntegerVector` vectors of type `integer`
• `NumericVector` vectors of type `numeric`
• `RawVector` vectors of type `raw`
• `LogicalVector` vectors of type `logical`
• `CharacterVector` vectors of type `character`
• `GenericVector` generic vectors implementing `list` types
Common core functions for Vectors and Matrices

Key operations for all vectors, styled after STL operations:

- `operator()` access elements via `()`
- `operator[]` access elements via `[]`
- `length()` also aliased to `size()`
- `fill(u)` fills vector with value of `u`
- `begin()` pointer to beginning of vector, for iterators
- `end()` pointer to one past end of vector
- `push_back(x)` insert `x` at end, grows vector
- `push_front(x)` insert `x` at beginning, grows vector
- `insert(i, x)` insert `x` at position `i`, grows vector
- `erase(i)` remove element at position `i`, shrinks vector
Basic Usage
evalCpp() evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```r
library(Rcpp)
evalCpp("2 + 2")  # simple test

## [1] 4

evalCpp("std::numeric_limits<double>::max()")

## [1] 1.797693e+308
```
**Basic Usage: cppFunction()**

cppFunction() creates, compiles and links a C++ file, and creates an R function to access it.

```cpp
cppFunction(""
    int exampleCpp11() {
        auto x = 10;
        return x;
    }", plugins=c("cpp11"))
exampleCpp11() # same identifier as C++ function
```
sourceCpp() is the actual workhorse behind evalCpp() and cppFunction()`. It is described in more detail in the package vignette Rcpp-attributes.

sourceCpp() builds on and extends cxxfunction() from package inline, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf RcppArmadillo, RcppEigen, RcppGSL).
Basic Usage: RStudio
The following file gets created:

```c++
#include <Rcpp.h>
using namespace Rcpp;

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). ...

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) { return x * 2; }

// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.

/*** R
timesTwo(42)
*/
```
So what just happened?

- We defined a simple C++ function
- It operates on a numeric vector argument
- We asked Rcpp to ‘source it’ for us
- Behind the scenes Rcpp creates a wrapper
- Rcpp then compiles, links, and loads the wrapper
- The function is available in R under its C++ name
Package are *the* standard unit of R code organization.

Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette `Rcpp-packages` has fuller details.

As of November 10, 2016, there are 832 packages on CRAN which use Rcpp, and a further 89 on BioConductor — with working, tested, and reviewed examples.
Best way to organize R code with Rcpp is via a package:
Rcpp.package.skeleton() and its derivatives. e.g. RcppArmadillo.package.skeleton() create working packages.

// another simple example: outer product of a vector, returning a matrix

//
// [[Rcpp::export]]
arma::mat rcpparma_outerproduct(const arma::colvec & x) {
    arma::mat m = x * x.t();
    return m;
}

// and the inner product returns a scalar

//
// [[Rcpp::export]]
double rcpparma_innerproduct(const arma::colvec & x) {
    double v = arma::as_scalar(x.t() * x);
    return v;
}
Two ways to link to external libraries

- *With linking of libraries*: Do what RcppGSL does and use hooks in the package startup to store compiler and linker flags, pass to environment variables

- *With C++ template headers only*: Do what RcppArmadillo and other do and just point to the headers

More details in extra vignettes.
Sugar Example
Syntactic ‘sugar’: Simulating $\pi$ in R

Draw $(x, y)$, compute dist $d$ to origin. Repeat. Ratio of points with $\sum I(d \leq 1)/N$ goes to $\pi/4$ as we fill the 1/4 of the unit circle.

```r
piR <- function(N) {
  x <- runif(N)
  y <- runif(N)
  d <- sqrt(x^2 + y^2)
  return(4 * sum(d <= 1.0) / N)
}
set.seed(5)
sapply(10^(3:6), piR)

```
Rcpp sugar enables us to write C++ code that is almost as compact.

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double piSugar(const int N) {
    NumericVector x = runif(N);
    NumericVector y = runif(N);
    NumericVector d = sqrt(x*x + y*y);
    return 4.0 * sum(d <= 1.0) / N;
}
```

The code is essentially identical.
Syntactic ‘sugar’: Simulating $\pi$

And by using the same RNG, so are the results.

```r
library(Rcpp)
sourceCpp("code/piSugar.cpp")
set.seed(42); a <- piR(1.0e7)
set.seed(42); b <- piSugar(1.0e7)
identical(a,b)
```

## [1] TRUE

```r
print(c(a,b), digits=7)
```

## [1] 3.140899 3.140899
The performance is close with a small gain for C++ as R is already vectorised:

```r
library(rbenchmark)
sourceCpp("code/piSugar.cpp")
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]
```

## test replications elapsed relative
## 1 piR(1e+06) 100 6.946 2.693
## 2 piSugar(1e+06) 100 2.579 1.000
Takeaways

• We can prototype in R to derive a first solution
• We can then rewrite performance-critical parts
• Key R functions are often available in C++ via Rcpp Sugar
• Random Number Simulation will work on identical streams
Other Examples
A basic looped version:

```cpp
#include <Rcpp.h>
#include <numeric> // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
    double acc = 0; // init an accumulator variable

    NumericVector res(x.size()); // init result vector

    for(int i = 0; i < x.size(); i++){
        acc += x[i];
        res[i] = acc;
    }

    return res;
}
```
An STL variant:

// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
    // initialize the result vector
    NumericVector res(x.size());
    std::partial_sum(x.begin(), x.end(), res.begin());
    return res;
}
Or just Rcpp sugar:

```cpp
// [[Rcpp::export]]
NumericVector cumsum_sug(NumericVector x){
    return cumsum(x);  // compute + return result vector
}
```

Of course, all results are the same.
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector callFunction(NumericVector x, Function f) {
    NumericVector res = f(x);
    return res;
}

/*** R
callFunction(x, fivenum)
*/
using Boost via BH: using-boost-with-bh

```cpp
// [[Rcpp::depends(BH)]]
#include <Rcpp.h>

// One include file from Boost
#include <boost/date_time/gregorian/gregorian_types.hpp>

using namespace boost::gregorian;

// [[Rcpp::export]]
Rcpp::Date getIMMDate(int mon, int year) {
    // compute third Wednesday of given month / year
    date d = nth_day_of_the_week_in_month(
        nth_day_of_the_week_in_month::third,
        Wednesday, mon).get_date(year);
    date::ymd_type ymd = d.year_month_day();
    return Rcpp::wrap(Rcpp::Date(ymd.year, ymd.month, ymd.day));
}
```
#include <Rcpp.h>
#include <boost/foreach.hpp>

using namespace Rcpp;

// [[Rcpp::depends(BH)]]

// the C-style upper-case macro name is a bit ugly
#define foreach BOOST_FOREACH

// [[Rcpp::export]]
NumericVector square( NumericVector x ) {
    // elem is a reference to each element in x
    // we can re-assign to these elements as well
    foreach( double& elem, x ) {
        elem = elem*elem;
    }
    return x;
}

C++11 now has something similar in a smarter for loop.
```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector positives(NumericVector x) {
    return x [ x > 0 ];
}

// [[Rcpp::export]]
List first_three(List x) {
    IntegerVector idx = IntegerVector::create(0, 1, 2);
    return x [ idx ];
}

// [[Rcpp::export]]
List with_names(List x, CharacterVector y) {
    return x [ y ];
}
```
#include <Rcpp.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
    return arma::eig_sym(M);
}
sourceCpp("code/armaeigen.cpp")

set.seed(42)
X <- matrix(rnorm(4*4), 4, 4)
Z <- X %*% t(X)
getEigenValues(Z)

## [,1]
## [1,] 0.3318872
## [2,] 1.6855884
## [3,] 2.4099205
## [4,] 14.2100108

# R gets the same results (in reverse)
# and also returns the eigenvectors.
#include <Rcpp.h>
using namespace Rcpp;

NumericVector createXts(int sv, int ev) {
    IntegerVector ind = seq(sv, ev);  // values

    NumericVector dv(ind);            // date(time)s == reals
dv = dv * 86400;                   // scaled to days
dv.attr("tzone") = "UTC";         // index has attributes
dv.attr("tclass") = "Date";

    NumericVector xv(ind);            // data has same index
    xv.attr("dim") = IntegerVector::create(ev-sv+1,1);
    xv.attr("index") = dv;
    CharacterVector cls = CharacterVector::create("xts","zoo");
xv.attr("class") = cls;
xv.attr(".indexCLASS") = "Date";
    // ... some more attributes ...

    return xv;
}
```cpp
#include "RcppMLPACK.h"

using namespace mlpack::kmeans;
using namespace Rcpp;

// [[Rcpp::depends(RcppMLPACK)]]

// [[Rcpp::export]]
List cppKmeans(const arma::mat& data, const int& clusters) {
    arma::Col<size_t> assignments;
    KMeans<> k;   // Initialize with the default arguments.
    k.Cluster(data, clusters, assignments);

    return List::create(Named("clusters") = clusters,
                        Named("result") = assignments);
}
```
Timing

Table 1: Benchmarking result

<table>
<thead>
<tr>
<th>test</th>
<th>replications</th>
<th>elapsed</th>
<th>relative</th>
<th>user.self</th>
<th>sys.self</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlKmeans(t(wine), 3)</td>
<td>100</td>
<td>0.028</td>
<td>1.000</td>
<td>0.028</td>
<td>0.000</td>
</tr>
<tr>
<td>kmeans(wine, 3)</td>
<td>100</td>
<td>0.947</td>
<td>33.821</td>
<td>0.484</td>
<td>0.424</td>
</tr>
</tbody>
</table>

Table taken 'as is' from RcppMLPACK vignette.
#include "RcppMLPACK.h"

using namespace Rcpp;
using namespace mlpack;
using namespace mlpack::neighbor;
using namespace mlpack::metric;
using namespace mlpack::tree;

// [[Rcpp::depends(RcppMLPACK)]]
// [[Rcpp::export]]
List nn(const arma::mat& data, const int k) {
  // using a test from MLPACK 1.0.10 file src/mlpack/tests/allknn_test.cpp
  CoverTree<LMetric<2>, FirstPointIsRoot,
           NeighborSearchStat<NearestNeighborSort> > tree =
    CoverTree<LMetric<2>, FirstPointIsRoot,
             NeighborSearchStat<NearestNeighborSort> >(data);

  NeighborSearch<NearestNeighborSort, LMetric<2>,
                CoverTree<LMetric<2>, FirstPointIsRoot,
                         NeighborSearchStat<NearestNeighborSort> > > coverTreeSearch;

  arma::Mat<size_t> coverTreeNeighbors;
  arma::mat coverTreeDistances;
  coverTreeSearch.Search(k, coverTreeNeighbors, coverTreeDistances);

  return List::create(Named("clusters") = coverTreeNeighbors,
                       Named("result") = coverTreeDistances);
MORE
• The package comes with eight pdf vignettes, and numerous help pages.
• The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp Stat & Data Anlys*)
• The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
• StackOverflow has a large collection of posts too.
• And a number of blog posts introduce/discuss features.
On sale since June 2013.
Appendix: If You Can’t Beat ’em
Content

• Single- or Multi-Language?
• Interlude
• Illustration
• Conclusion
SINGLE- OR MULTI-LANGUAGE?
Better with more than one?

- No one language fits all
- Real-world projects are frequently multi-language
- See *e.g.* job ads which rarely ever list just one language
Counter-claim: 1 + 1 < 2

Or better with just one?

- Mental switching cost between languages? Possibly
- Interop difficult and less portable? Maybe, but that is an argument against weak systems / OSs
- Easier / less to learn?
- “More hoops” to code?
Mental switching costs?

Dirk Eddelbuettel
@eddelbuettel

Forcing us to alternate between comment characters %, # and // may have been the biggest trick ever pulled by the Devil.

10:22 AM - 10 Aug 2014
Open Question

- Hard to measure or test: Any empirics on real world projects?
- Code competition / comparisons (e.g. Project Euler): Are they realistic?
INTERLUDE
Chambers (2008) Software For Data Analysis
Chapters 10 and 11 devoted to Interfaces I: C and Fortran and Interfaces II: Other Systems.
Chambers (2016) Extending R
An entire book about this with *concrete* Python, Julia and C++ code and examples
Chambers 2016, Chapter 1

• *Everything that exists in R is an object*

• *Everything happens in R is a function call*

• *Interfaces to other software are part of R*
The fundamental lesson about programming in the large is that requires a correspondingly broad and flexible response. In particular, no single language or software system is likely to be ideal for all aspects. Interfacing multiple systems is the essence. Part IV explores the design of interfaces from R.
ILLUSTRATION
Setup

```r
py_cflags <- system("python2.7-config --cflags", intern=TRUE)
se <- Sys.setenv; ge <- Sys.getenv # shorthands to typeset
se("PKG_CFLAGS"=sprintf("%s %s", ge("PKG_CFLAGS"), py_cflags))
se("PKG_CXXFLAGS"=sprintf("%s %s", ge("PKG_CXXFLAGS"), py_cflags))
py_ldflags <- system("python2.7-config --ldflags", intern=TRUE)
se("PKG_LIBS"=sprintf("%s %s %s", ge("PKG_CFLAGS"),
    "-lboost_python-py27", py_ldflags))
```
#include <Rcpp.h>
#include <Python.h>

// [[Rcpp::export]]
void initialize_python() {
  Py_SetProgramName(""); /* optional but recommended */
  Py_Initialize();
}

// [[Rcpp::export]]
void hello_python() {
  PyRun_SimpleString("from time import time,ctime\n"  
  "print 'Today is',ctime(time())\n");
}
Hello, World: Called from R

```r
initialize_python()
hello_python()
```

## Today is Thu Nov 10 09:40:26 2016


Disclaimer: For illustration purposes. Works as designed on Ubuntu. Not meant to be universally portable to all three OSs.
(Section) Conclusion
Mixing Languages

- Common
- Natural
- Unavoidable
Consequences

• Must make it easier to interoperate
• Stop bickering among ourselves
• Build systems that are larger that the sum of their parts
Just Do It

Extending R

John M. Chambers

The R Series

UIUC Nov 2016
Lars Wirzenius “Which license is the most free?”

Free software licences can be roughly grouped into permissive and copyleft ones. [...] A permissive licence lets you do things that a copyleft one forbids, so clearly the permissive licence is more free. A copyleft licence means software using it won’t ever become non-free against the wills of the copyright holders, so clearly a copyleft licence is more free than a permissive one.

Both sides are both right and wrong, of course, which is why this argument will continue forever. [...] If a discussion about the relative freedom of licence types becomes heated, step away. It’s not worth participating anymore.

http://yakking.branchable.com/posts/comparative-freeness/
APPENDIX: OPEN SOURCE FINANCE
Issues

• History: How did we get here?
• Status: What is happening now
• Onward: What may happen
To clarify

- This talk reflects views of a quantitative analyst
- *Software* to us is predominantly a collection of analysis and modeling tools including programming languages, libraries, OSs
- The focus is on *Open Source Finance* — and much less about Open Source and Software in general
- Insert your favourite disclaimer here
HISTORY
Terms and Players

- *Open Source* dominates commercial discussions
- *Free Software* predates it; academic roots / MIT
- past friction between sponsoring entities
- OSI and FSF are closer now
Free as in the Freedom to ...

- run the program as you wish, for any purpose
- study how the program works, and change it
- redistribute copies so you can help your neighbor
- distribute copies of your modified versions to others

Access to source code is a precondition
Free Software and the GNU Public License (GPL)

GPL: A key Free Software License

• ‘Copyleft’: right to freely distribute copies and modified versions
• Stipulates that the same rights be preserved in derivative works
• ‘Viral’: Combined works have same (aggregate) license
• Some claim that this is not ‘permissive’
BSD/MIT/Apache Licenses

- These license calls themselves ‘more permissive’ – ie not viral
- Allows re-use and re-licensing: “can be taken private”
- One way to think about this is
  - **user-focus** of GPL: nobody can ever take current (or future versions) away
  - **author-focus** of BSD/MIT as not limiting (?) deployment

Perceived “conflict” overblown – both are Open Source licenses
“It’s complicated”

- This gets into ‘need a lawyer’ territory real fast
- Good (neutral) website: http://tldrlegal.com
- Main thing: Just pick any good recognized license
Key Aspects

• Focus on Software: ‘Infinitely copyable’
• Consider recent ‘newsworthy’ software releases (e.g. TensorFlow)
• ‘Open by Default’ a (related) winning concept:
  • Wikipedia
  • GitHub
Open Source Has Won

For Software, Debate is Over

- From Ballmer’s Microsoft: Linux is a Cancer
- To Nadella’s Microsoft: We love Linux
- Today, few areas of the software industry remain unchanged
- Now frequently seen: ‘Open Core’ base with add-on services
Microsoft embracing R

Openness

First off, Microsoft's embrace of open source is now a fact, rather than an issue. The company gets that open source platforms are de facto industry standards, and that customers like products that support them. Microsoft already has a version of HDInsight, its Big Data platform based on open source Hadoop and Spark technologies, that runs on Linux. It is also developing a version of SQL Server itself for Linux. Then there's Visual Studio Code, which runs on Windows, Mac or Linux. And a large portion of the virtual machines in the Microsoft Azure cloud are running Linux too.

Source: http://www.zdnet.com/article/microsofts-r-strategy/ (retrieved on 2016-May-14)
TRADING AND TRADING FIRMS
So Where Does that leave us?

Status Quo Somewhat Obvious and Boring

- Open Source is simply how software is done / used
- Trading / Wall St have used Open Source since forever
- Niche applications with premiums remain closed
  - As do ‘aggregations’ and OSs
  - OS X, Windows, ... as well, but at lower prices
- Hence: ‘Default is Open’
- I.e. last relevant + closed source programming language?
AN UPDATE IS AVAILABLE FOR YOUR COMPUTER

Cool, more free stuff!

Not again!

Ooh, only $99!

Linux

Windows

Mac

Source: http://www.stickycomics.com/computer-update/
So Yes, It is 80/20

Open Source Is

- what you use for your (scripting) languages
- what you use for your domain language
- what you use for your (No-)SQL backends
- and on and on and on

UIUC Nov 2016
Leaves Focus on Value-Added

- Strategies
- Analysis
- Core (in-house) Technology
to differentiate
PARTICIPATE
Why?

Signalling!

- Better hiring
- Better staff morale
- Better code
WHO PLAYS?

A very incomplete list

- TwoSigma Beaker Notebook
- Bloomberg via
  - large C++ libraries
  - OpenBloomberg API libraries
- Goldman Sachs Java Collections Framework
BEAKER
THE DATA SCIENTIST'S LABORATORY

Beaker is a notebook-style development environment for working interactively with large and complex datasets. Its plugin-based architecture allows you to switch between languages or add new ones with ease, ensuring that you always have the right tool for any of your analysis and visualization needs.

Get Beaker
Mac | Win | Linux

Run Beaker
Cloud Hosted

The Perfect Tool for Iterative Exploration
Main Issue:

- Finance / Trading **not** known as a supporter / contributor
  - I.e. Morgan Stanley employs Stroustrup
  - But e.g. why is van Rossum not employed in the industry?
  - Not aware of other key OS developers employed
- But could this be changing?
**Small Steps**

- **UseR! 2016** co-sponsored by RenTec, TwoSigma, Bridgewater
- Ketchum has sponsored NIPS, R/Finance and R Consortium
- Funding opportunities:
  - R now has the **R Consortium**
  - Python (et al) have **NumFocus**
  - Linux has the **Linux Foundation**
- But also
  - **Software Freedom Conservancy**
  - **Software in the Public Interest**
Industry distribution

Based on the C/C++ job ads we analyzed, C++ is most used in industries like Finance and Banking.


With thanks to Michael Wong and his STAC Chicago presentation on May 17, 2016.
Trading

- Benefits hugely as a ‘shadow IT industry’
- By and large does not seem to contribute back
- Let’s try to change that
One More Thing
Software Carpentry (and Data Carpentry)

- Basic shell skills
- Basics of version control
- Good programming practice (R, Python, Matlan, ...)

are essential for today’s students and tomorrow’s researchers
CONCLUDING
Key Themes

- Statistics largely computational
- R is a key ingredient
- Rcpp is a performant and expressive API extension
- Extending R is a key feature
  - Programming is (often) multi-lingual
  - Extending to other systems / languages natural
- Open Source is a key aspect
- Important to teach more than just single language
Thank You!

http://dirk.eddelbuettel.com/
dirk@eddelbuettel.com
@eddelbuettel