



AN INTRODUCTION TO RCPP

A HANDS-ON OVERVIEW

Dirk Edelbuettel

NISS Virtual Seminar

25 Mar 2022

https://dirk.edelbuettel.com/papers/niss_rcppIntro_mar2022.pdf

WHO AM I ?

[tile]DB

Products ▾

Blog

Documentation

Community ▾

Log in

Sign up

Data management made universal

Manage any data as multi-dimensional arrays and access with any tool at global scale.

Get started

Request a demo



(Adjunct) Clinical Professor, University of Illinois

- teaching [STAT447 'Data Science Programming Methods'](#)

Open Source Work

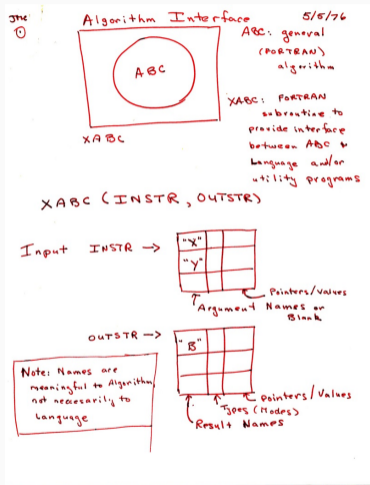
- Debian developer since 1995, currently maintaining about 185 packages
- R package author since 2002, author or maintainer of over 60 CRAN packages
- R Foundation Board Member; JSS Associate Editor
- Rocker Project co-founder: Docker for R, including official 'r-base' image

INTRODUCTION TO RCPP

Overview

- Some Historical Context (or “Why?”)
- Brief Notes on First Steps (or “How?”)
- Some Tips and Further Reference
- Interwoven with Some Empirics

HISTORY: BELL LABS, 1976



Bell Labs meeting notes from May 1976 (!!)

Describes an outer 'user interface' layer to algorithms, with extensibility a core feature

Key: abstract away inner details giving high-level accessible view for analyst

Became "The Interface" / "The System" or just "S"

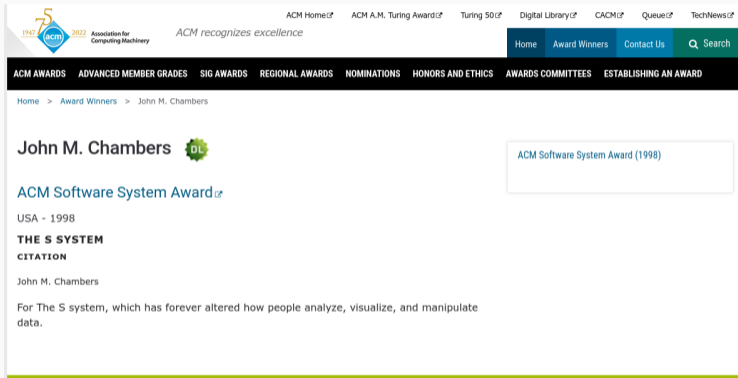
Which in turn became S which lead to R

So see R as an interactive shell, interpreter and language over numeric, statistical, i/o, ... ops

A language by statisticians for statisticians

Source: Chambers (2008, Appendix A); Chambers (2020), *S, R, and Data Science*, Proc ACM Prog Lang, Vol 4, Issue HOPL, doi.org/10.1145/3386334.

IMPACT: ACM SOFTWARE SYSTEMS AWARD, 1998



The screenshot shows the ACM website's award page for John M. Chambers. The page header includes the ACM logo (1947-2022) and the tagline "ACM recognizes excellence". Navigation links include "ACM Home", "ACM A.M. Turing Award", "Turing 50", "Digital Library", "CACM", "Queue", and "TechNews". A secondary navigation bar contains "Home", "Award Winners", "Contact Us", and "Search". A third navigation bar lists categories: "ACM AWARDS", "ADVANCED MEMBER GRADES", "SIG AWARDS", "REGIONAL AWARDS", "NOMINATIONS", "HONORS AND ETHICS", "AWARDS COMMITTEES", and "ESTABLISHING AN AWARD". The breadcrumb trail reads "Home > Award Winners > John M. Chambers". The main content area features the name "John M. Chambers" with a "DL" icon, the award title "ACM Software System Award", the location and year "USA - 1998", the award name "THE S SYSTEM", and the citation: "For The S system, which has forever altered how people analyze, visualize, and manipulate data." A box on the right side of the page contains the text "ACM Software System Award (1998)".

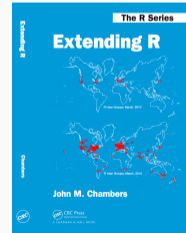
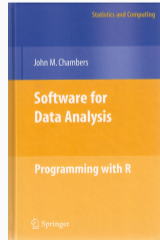
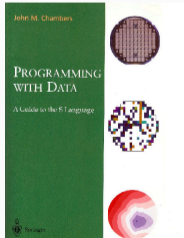
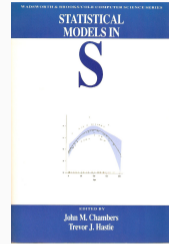
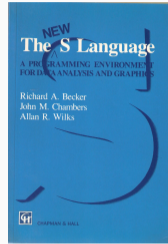
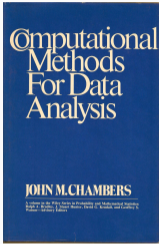
For The S System, which has forever altered how people analyze, visualize, and manipulate data.

Other Winners:

BerkeleyDB (2020), DNS (2019),
Wireshark (2018), Project
Jupyter (2017), GCC (2015), LLVM
(2012), VMware (2009), Eiffel
(2006), make (2003), Java (2002),
Apache (1999), Tcl/Tk (1997),
NCSA Mosaic (1996), World Wide
Web (1995), Remote Procedure
Call (1994), TCP/IP (1991),
Postscript (1989), Smalltalk
(1987), TeX (1986), VisiCalc
(1985), UNIX (1983)

Source: https://en.wikipedia.org/wiki/ACM_Software_System_Award

EVOLUTION OF PROGRAMMING WITH DATA FROM 1977 TO 2016



Chambers (2016, pages 4-11)

objects: Everything that exists in R is an object.

functions: Everything that happens in R is a function call.

interfaces: Interfaces to other languages are a part of R.

INTRODUCTION: WHY DO THIS?

Three key reasons

- Speed, Performance, ...
- Do things you could not do before
- Easy to extend R this way

Why Choose 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 rich inheritance and interoperability: Unix, Windows, ... all built 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, C++14, C++17, C++20* (and beyond) add enough to be called a fifth language.

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

Reasons For 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, C++17, C++20, ... are a big deal giving us new language features
- While there are (plenty of) complexities, Rcpp users are somewhat shielded

GETTING STARTED

R Version of 'is this number odd or even'

```
isOdd_r <- function(num = 10L) {  
  result = (num %% 2L == 1L)  
  return(result)  
}  
c(isOdd_r(42L), isOdd_r(43L))
```

```
## [1] FALSE TRUE
```

SIMPLE EXAMPLE (CONT.)

C++ Version of 'is this number odd or even'

```
bool isOdd_cpp(int num = 10) {  
    bool result = (num % 2 == 1);  
    return result;  
}
```

Free-standing code, not yet executable...

Rcpp Version of 'is this number odd or even'

```
Rcpp::cppFunction("
bool isOdd_cpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}")
c(isOdd_r(42L), isOdd_r(43L))
```

```
## [1] FALSE TRUE
```

SIMPLE EXAMPLE (CONT.)

In R

```
##  
isOdd_r <- function(n=10L) {  
  res = (n %% 2L == 1L)  
  return(res)  
}  
isOdd_r(42L)
```

```
## [1] FALSE
```

In C++ via Rcpp

```
Rcpp::cppFunction("  
bool isOdd_cpp(int n=10) {  
  bool res = (n % 2 == 1);  
  return res;  
}")  
isOdd_cpp(42L)
```

```
## [1] FALSE
```

SECOND EXAMPLE: VAR(1)

Let's consider a simple possible VAR(1) system of k variables.

For $k = 2$:

$$X_t = X_{t-1}B + E_t$$

where X_t is a row vector of length 2, B is a 2 by 2 matrix, and E_t is a row of the error matrix of 2 columns.

SECOND EXAMPLE: VAR(1)

In R code, given both the coefficient and error matrices (revealing k and n):

```
rSim <- function(B,E) {  
  X <- matrix(0,nrow(E), ncol(E))  
  for (r in 2:nrow(E)) {  
    X[r,] = X[r-1, ] %*% B + E[r, ]  
  }  
  return(X)  
}
```

SECOND EXAMPLE: VAR(1)

```
Rcpp::cppFunction('arma::mat cppSim(arma::mat B, arma::mat E) {
  int m = E.n_rows, n = E.n_cols;
  arma::mat X(m,n);
  X.row(0) = arma::zeros<arma::mat>(1,n);
  for (int r=1; r<m; r++)
    X.row(r) = X.row(r-1) * B + E.row(r);
  return X; }', depends="RcppArmadillo")
a <- matrix(c(0.5, 0.1, 0.1, 0.5), nrow=2)
e <- matrix(rnorm(10000), ncol=2)
rbenchmark::benchmark(cppSim(a,e), rSim(a,e), order="relative")[,1:4]
```

```
##           test replications elapsed relative
## 1 cppSim(a, e)           100   0.010         1.0
## 2  rSim(a, e)           100   0.728        72.8
```

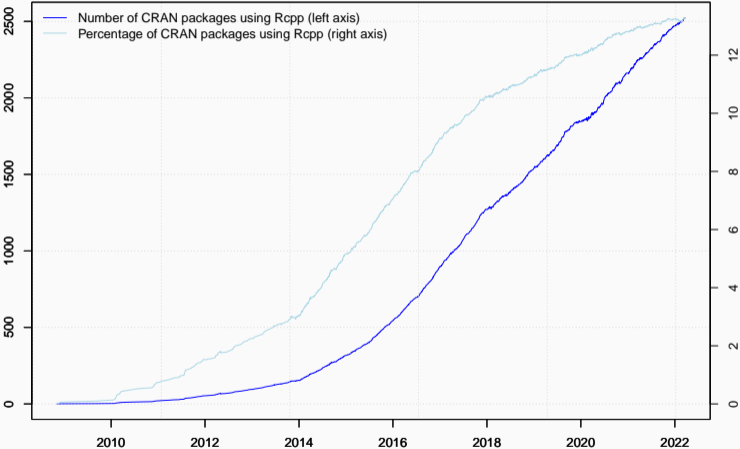
New things: Easy access to C/C++ libraries

- Sometimes speed is not the only reason
- C & C++ provide numerous libraries + APIs we may want to use
- Easy to provide access to as Rcpp eases data transfer

EMPIRICAL ASIDE

GROWTH OF RCPP

Growth of Rcpp usage on CRAN



Data current as of March 21, 2022.

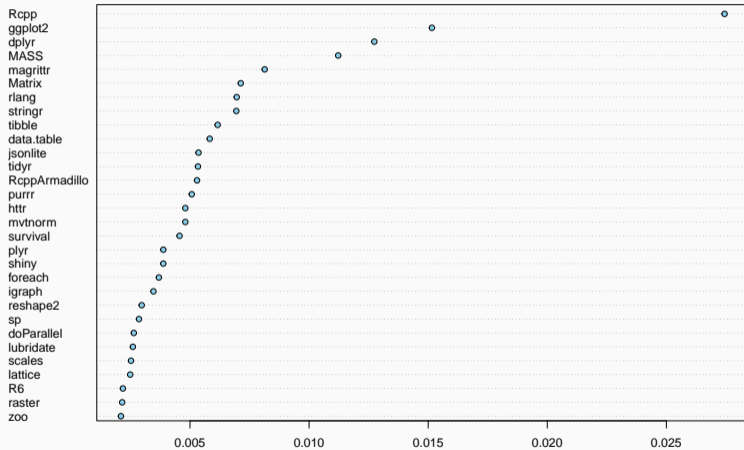
Rcpp is currently used by

- 2524 CRAN packages
- 239 BioConductor packages
- an unknown (but “large”) number of GitHub projects

```
suppressMessages(library(utils))  
library(pagerank) # cf github.com/andrie/pagerank  
  
cran <- "https://cloud.r-project.org"  
pr <- compute_pagerank(cran)  
round(100*pr[1:5], 3)
```

```
##      Rcpp  ggplot2    dplyr     MASS magrittr  
##      2.744   1.516   1.274   1.122   0.814
```

Top 30 of Page Rank as of March 2022



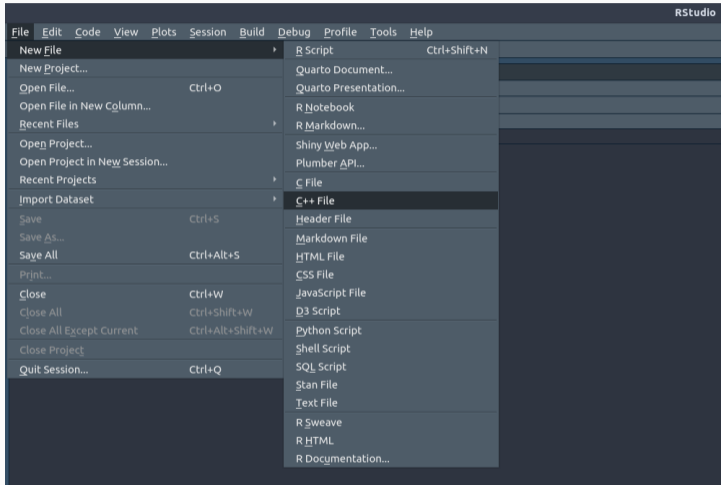
PERCENTAGE OF COMPILED PACKAGES

```
db <- tools::CRAN_package_db() # added in R 3.4.0
db <- db[!duplicated(db[,1]),] # rows: nb of pkgs,
nTot <- nrow(db) # cols: diff attributes
nRcpp <- length(tools::dependsOnPkgs("Rcpp",recursive=FALSE, installed=db))
nCompiled <- table(db[, "NeedsCompilation"])[["yes"]]
propRcpp <- nRcpp / nCompiled * 100
data.frame(tot=nTot, totRcpp = nRcpp, totCompiled = nCompiled,
           RcppPctOfCompiled = propRcpp)
```

```
##      tot totRcpp totCompiled RcppPctOfCompiled
## 1 19004   2524      4462          56.5666
```

INTRODUCTION: HOW TO DO THIS?

JUMPING RIGHT IN: VIA RSTUDIO



A FIRST EXAMPLE: CONT'ED

```
#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 ask 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

Consider a function defined as

$$f(n) \text{ such that } \begin{cases} n & \text{when } n < 2 \\ f(n-1) + f(n-2) & \text{when } n \geq 2 \end{cases}$$

AN INTRODUCTORY EXAMPLE: SIMPLE R IMPLEMENTATION

R implementation and use:

```
f <- function(n) {  
  if (n < 2) return(n)  
  return(f(n-1) + f(n-2))  
}  
sapply(0:10, f) # Using it on first 11 arguments
```

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

AN INTRODUCTORY EXAMPLE: TIMING R IMPLEMENTATION

Timing:

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

##	test	replications	elapsed	relative
## 1	f(10)	100	0.008	1.000
## 2	f(15)	100	0.097	12.125
## 3	f(20)	100	1.101	137.625

AN INTRODUCTORY EXAMPLE: C++ IMPLEMENTATION

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

deployed as

```
Rcpp::cppFunction('int g(int n) {  
    if (n < 2) return(n);  
    return(g(n-1) + g(n-2)); }')  
sapply(0:10, g) # Using it on first 11 arguments
```

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

AN INTRODUCTORY EXAMPLE: COMPARING TIMING

Timing:

```
library(rbenchmark)
benchmark(f(20), g(20))[,1:4]
```

```
##      test replications elapsed relative
## 1 f(20)           100    1.034      517
## 2 g(20)           100    0.002         1
```

A nice gain of a few orders of magnitude.

SOME BACKGROUND

R Type mapping

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works:

```
library(Rcpp)
cppFunction("NumericVector logabs(NumericVector x){
  return log(abs(x));
}")
logabs(seq(-5, 5, by=2))
```

Also note: vectorized C++! Here `log(abs())` runs directly on vectors as R would.

STL TYPE MAPPING

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

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(), f);
    return x;
}
```

Not vectorized but 'sweeps' `f()` along `std::vector<double>` via STL `std::transform()`

Used via

```
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))
```

TYPE MAPPING IS SEAMLESS

Simple outer product of a col. vector (using RcppArmadillo):

```
Rcpp::cppFunction("arma::mat v(arma::colvec a) { return a*a.t(); }",  
                  depends="RcppArmadillo")  
  
v(1:3)
```

```
##      [,1] [,2] [,3]  
## [1,]    1    2    3  
## [2,]    2    4    6  
## [3,]    3    6    9
```

Uses implicit conversion via `as<>` and `wrap` – cf [vignette Rcpp-extending](#).

We can simplify the `log(abs(...))` example further:

```
#include <Rcpp.h>
// [[Rcpp::plugins(cpp11)]]

using namespace Rcpp;

// [[Rcpp::export]]
std::vector<double> logabs3(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(),
                  [](double x) { return ::log(::fabs(x)); } );
    return x;
}
```

HOW TO: MAIN USAGE PATTERNS

BASIC USAGE: EVALCPP()

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

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

```
## [1] 4
```

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

```
## [1] 1.79769e+308
```

BASIC USAGE: CPPFUNCTION()

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

```
cppFunction("
  int exampleCpp11() {
    auto x = 10;
    return x;
}", plugins=c("cpp11"))
exampleCpp11() # same identifier as C++ function
```


BASIC USAGE: SOURCECPP()

`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`).

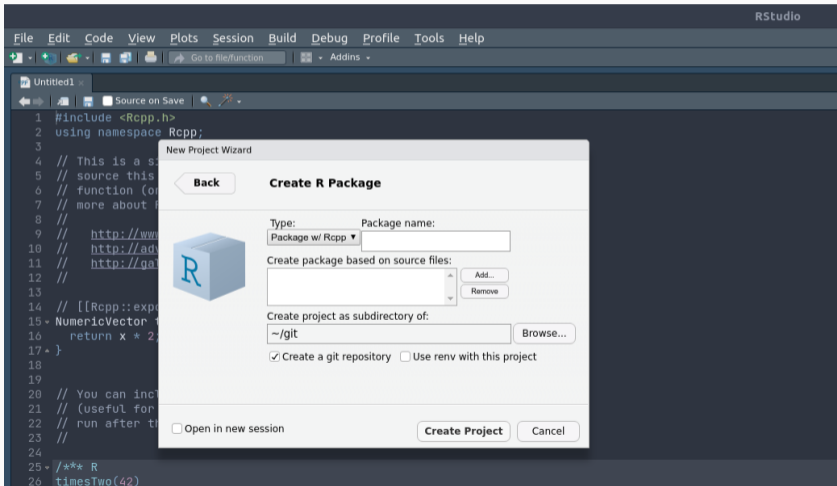
Package are *the* standard unit of R code organization.

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 March 2022, there are 2524 CRAN and 239 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

Best way to organize R code with Rcpp is via a package:



Rcpp.package.skeleton() and 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 (or three) ways to link to external libraries

- *Full copies*: Do what `mlpack` does and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries*: Do what e.g. `RcppGSL` does and use hooks in the package startup to store compiler and linker flags which are passed 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.

Vignette and pre-print

The screenshot shows a web browser window displaying an arXiv pre-print page. The browser's address bar shows the URL `arxiv.org/abs/1911.06416`. The page header includes the Cornell University logo and a search bar. The main content area is titled "Thirteen Simple Steps for Creating An R Package with an External C++ Library" by Dirk Eddelbuettel. The text describes extending R with an external C++ code library using the Rcpp package. The page includes sections for "Subjects", "Bibliographic data", and "Submission history". A right-hand sidebar contains "Download:" options (PDF, PostScript, Other formats), "Current browse context:", "References & Citations", and "Bookmark".

[1911.06416] Thirteen Simple Steps for Creating An R Package with an External C++ Library - Google Chrome

arXiv.org/abs/1911.06416

Cornell University

We gratefully acknowledge support from the Simons Foundation and member institutions.

arXiv.org > stat > arXiv:1911.06416

Search... All fields Search

Help | Advanced Search

Statistics > Computation

[Submitted on 14 Nov 2019]

Thirteen Simple Steps for Creating An R Package with an External C++ Library

Dirk Eddelbuettel

We describe how we extend R with an external C++ code library by using the Rcpp package. Our working example uses the recent machine learning library and application 'Corels' providing optimal yet easily interpretable rule lists <arXiv:1704.01701> which we bring to R in the form of the 'RcppCorels' package. We discuss each step in the process, and derive a set of simple rules and recommendations which are illustrated with the concrete example.

Subjects: **Computation** [stat.CO]

Cite as: arXiv:1911.06416 [stat.CO]
(or arXiv:1911.06416v1 [stat.CO] for this version)

Bibliographic data

[Enable Bibex (What is Bibex?)]

Submission history

From: Dirk Eddelbuettel [view email]
[v1] Thu, 14 Nov 2019 23:42:35 UTC (24 KB)

[Which authors of this paper are endorsers?](#) | [Disable MathJax \(What is MathJax?\)](#)

Download:

- PDF
- PostScript
- Other formats

(source)

Current browse context:

stat.CO

< prev | next >
new | recent | 1911

Change to browse by:

stat

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

Export citation

Bookmark

🔖 📄 📄 📄

BIG PICTURE

Choice is yours

- Code generation helps remove tedium
- Interfaces are shorter / simpler / more R like
 - recall the `is_odd` function earlier
- Plain C API to R is of course *perfectly fine*
- But IMHO requires **more work**
 - more manual steps for type conversion
 - additional required memory protection
 - all of which is **error prone**

COMPARE

```
#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;
}
```

```
#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);
}
```

You always have a choice between the code (from Section 5.10.1 of *Writing R Extensions*) on the left, or the equivalent Rcpp code on the right.

THIRD EXAMPLE: SUGAR

SYNTACTIC 'SUGAR': SIMULATING π IN R

Draw (x, y) , compute distance to origin.

Do so repeatedly, and ratio of points below one to number N of simulations will approach $\pi/4$ as we fill the area of $1/4$ of the unit circle.

```
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)
```

```
## [1] 3.15600 3.15520 3.13900 3.14101
```

SYNTACTIC 'SUGAR': SIMULATING π IN C++

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

The code is essentially identical.

```
#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;
}
```

SYNTACTIC 'SUGAR': SIMULATING π

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

```
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
```

```
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:

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

##		test	replications	elapsed	relative
## 1	piR(1e+06)		100	4.606	2.938
## 2	piSugar(1e+06)		100	1.568	1.000

TIPS

AN (ALMOST) FREQUENTLY ASKED QUESTION

Compare These Two Cases

```
## implicit pass by value
```

```
Rcpp::cppFunction("void impl(NumericVector X) {  
    X = X + 1.0; }")
```

```
a <- 1.5:4.5
```

```
impl(a)
```

```
a
```

```
## [1] 2.5 3.5 4.5 5.5
```

```
## implicit pass by reference (via &)
```

```
Rcpp::cppFunction("void expl(NumericVector& X) {  
    X = X + 1.0; }")
```

```
b <- 1.5:4.5
```

```
expl(b)
```

```
b
```

```
## [1] 2.5 3.5 4.5 5.5
```

The *implicit* case will surprise C++ programmers as they expect ‘pass by value’ semantics. But we have *SEXP* here: Pointers!

See Rcpp FAQ Question 5.1 for longer discussion.

ANOTHER (ALMOST) FREQUENTLY ASKED QUESTION

Now Compare These Two Cases

passing an integer vector

```
Rcpp::cppFunction("void intvec(IntegerVector X) {  
    X = X + 1.0; }")
```

```
a <- 1:5
```

```
intvec(a)
```

```
a # changed as side effect
```

```
## [1] 2 3 4 5 6
```

passing a numeric vector

```
Rcpp::cppFunction("void numvec(NumericVector& X) {  
    X = X + 1.0; }")
```

```
b <- 1:5
```

```
numvec(b)
```

```
b # unchanged -- why?
```

```
## [1] 1 2 3 4 5
```

The `int` vector needed to get copied to `numeric` in the second case, and that `numeric` vector is not connected to the `int` vector.

See the second part of Rcpp FAQ Question 5.1 for more.

Careful with Function Arguments

- Best to consider them 'read-only' and not assign to them
- Difficult to make them 'non-mutable' (without changing existing Rcpp semantics)
- Keep these examples in mind:
 - you may get changes in the input argument even if you think 'by value'
 - or you may 'accidentally' not get changes because of a cast / copy

Can Be Challenging

- We generally lack good *integrated* tooling for integrated R *and* C++ debugging
- Some tutorials exist, current favourite of some users is VS Code
- “Old-school” **gdb** and alike work (with some getting used-to)
- “Old-school” **print** statements and alike also work (albeit inelegantly)

MORE

Core Documentation

- The package comes with ten pdf vignettes, and help pages.
- The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp Stat & Data Anlys*, Rcpp again in *TAS*)
- The rcpp-devel mailing list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has over 2800 posts on Rcpp as well (and is searchable).
- And a number of blog posts introduce/discuss features.

Rcpp Gallery - Google Chrome

Rcpp Gallery x

gallery.rcpp.org

Rcpp Projects - Gallery Book Events More -

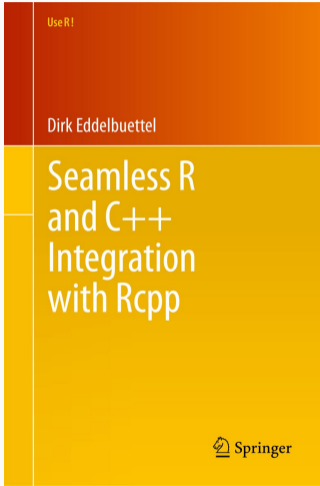
Featured Articles

- [Quick conversion of a list of lists into a data frame](#) — John Merrill
This post shows one method for creating a data frame quickly
- [Passing user-supplied C++ functions](#) — Dirk Eddelbuettel
This example shows how to select user-supplied C++ functions
- [Using Rcpp to access the C API of xts](#) — Dirk Eddelbuettel
This post shows how to use the exported API functions of xts
- [Timing normal RNGs](#) — Dirk Eddelbuettel
This post compares drawing $N(0,1)$ vectors from R, Boost and C++11
- [A first lambda function with C++11 and Rcpp](#) — Dirk Eddelbuettel
This post shows how to play with lambda functions in C++11
- [First steps in using C++11 with Rcpp](#) — Dirk Eddelbuettel
This post shows how to experiment with C++11 features
- [Using Rcout for output synchronised with R](#) — Dirk Eddelbuettel
This post shows how to use Rcout (and Rcerr) for output
- [Using the Rcpp sugar function clamp](#) — Dirk Eddelbuettel
This post illustrates the sugar function clamp
- [Using the Rcpp Timer](#) — Dirk Eddelbuettel
This post shows how to use the Timer class in Rcpp
- [Calling R Functions from C++](#) — Dirk Eddelbuettel
This post discusses calling R functions from C++

[More »](#)

Recently Published

- Apr 12, 2013 » [Using the RcppArmadillo-based Implementation of R's sample\(\)](#) — Christian Gunning and Jonathan Olmsted
- Apr 8, 2013 » [Dynamic Wrapping and Recursion with Rcpp](#) — Kevin Ushey
- Mar 14, 2013 » [Using bigmemory with Rcpp](#) — Michael Kane
- Mar 12, 2013 » [Generating a multivariate gaussian distribution using RcppArmadillo](#) — Ahmadou Dicko
- Mar 1, 2013 » [Using Rcpp with Boost.Regex for regular expression](#) — Dirk Eddelbuettel
- Feb 27, 2013 » [Fast factor generation with Rcpp](#) — Kevin Ushey



On sale since June 2013.

THANK YOU!

slides <https://dirk.eddelbuettel.com/presentations/>

web <https://dirk.eddelbuettel.com/>

mail dirk@eddelbuettel.com

github [@eddelbuettel](#)

twitter [@eddelbuettel](#)