

EXTENDING R

A BRIEF OVERVIEW

Dirk Eddebuettel

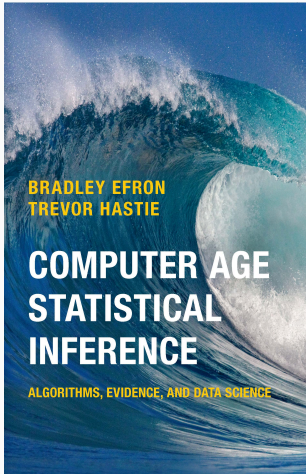
Northwestern University R Users Group

May 22, 2019

Extending R

- Why R ?
- Why Extending R ?
- Why Rcpp ?
- (Briefly) How ?

WHY R: VIEW FROM ACADEMIA



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

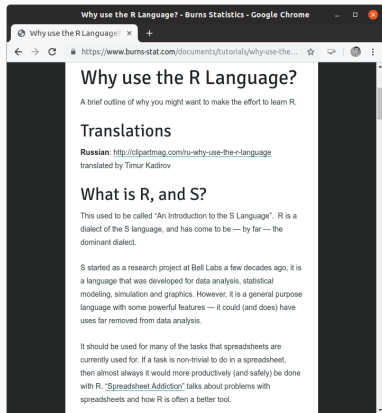
pages xv and 6 (footnote 3)

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)

WHY R: VIEW FROM PRACTITIONERS

WHY R? PAT BURN'S VIEW



Why the R Language?

Short essay at [Burns-Stat](#)

His site has more truly excellent (and free) writings.



Why the R Language?

- R is not just a statistics package, it's a language.
- R is designed to operate the way that problems are thought about.
- R is both flexible and powerful.

Source: <https://www.burns-stat.com/documents/tutorials/why-use-the-r-language/>



Why R for data analysis?

R is not the only language that can be used for data analysis. Why R rather than another? Here is a list:

- interactive language
- data structures
- graphics
- missing values
- functions as first class objects
- packages
- community

Source: <https://www.burns-stat.com/documents/tutorials/why-use-the-r-language/>

WHY R: PROGRAMMING WITH DATA

R as an Extensible Environment

- As R users we know that R can
 - ingest data in many formats from many sources
 - aggregate, slice, dice, summarize, ...
 - visualize in many forms, ...
 - model in just about any way
 - report in many useful and scriptable forms
- It has become central for **programming with data**
- Sometimes we want to extend it further than R code goes



From any one of

- csv
- txt
- xlsx
- xml, json, ...
- web scraping, ...
- hdf5, netcdf, ...
- sas, stata, spss, ...
- various SQL + NOSQL DBs
- various binary protocols

via

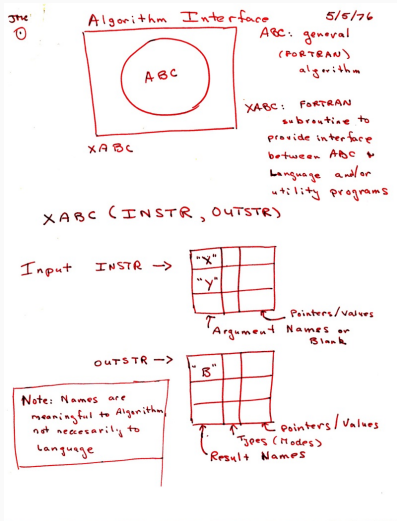


into any one of

- txt
- html
- latex and pdf
- html and js
- word
- shiny
- most graphics formats
- other dashboards
- web frontends

WHY R: HISTORICAL PERSPECTIVE

R AS 'THE INTERFACE'



A design sketch called 'The Interface'

AT&T Research lab meeting notes

Describes an outer 'user interface' layer to core Fortran algorithms

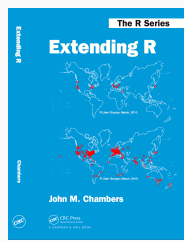
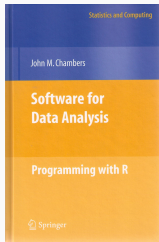
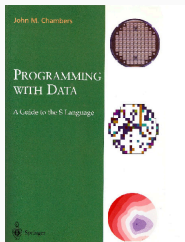
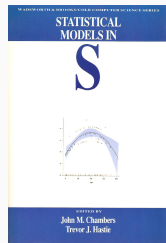
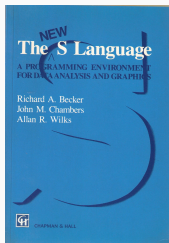
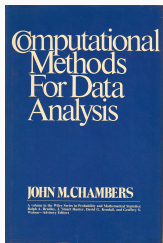
Key idea of abstracting away inner details giving higher-level more accessible view for user / analyst

Lead to "The Interface"

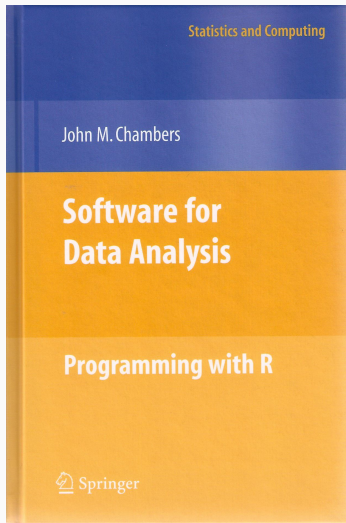
Which became S which lead to R

Source: John Chambers, personal communication

WHY R? : PROGRAMMING WITH DATA FROM 1977 TO 2016

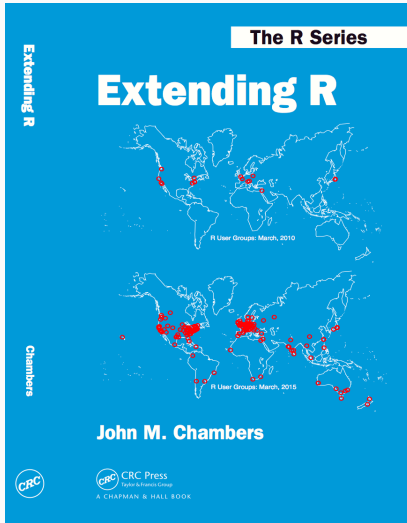


Thanks to John Chambers for high-resolution cover images. The publication years are, respectively, 1977, 1988, 1992, 1998, 2008 and 2016.



Software For Data Analysis

Chapters 10 and 11 devoted to *Interfaces I: C and Fortran* and *Interfaces II: Other Systems*.

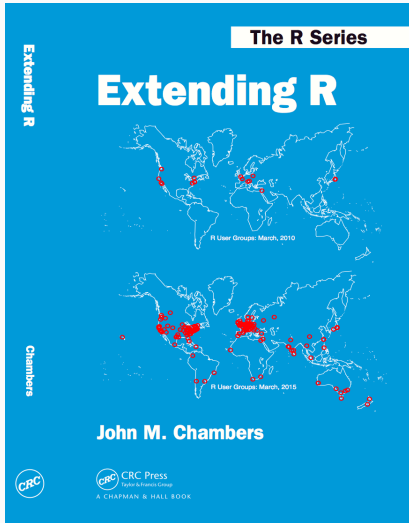


Extending R

Object: Everything that exists in R is an object

Function: Everything happens in R is a function call

Interface: Interfaces to other software are part of R



Extending R, Chapter 4

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.

A good fit, it turns out

- A good part of R is written in C (besides R and Fortran code)
- The principle interface to external code is a function `.Call()`
- It takes one or more of the high-level data structures R uses
- ... and returns one. Formally:

```
SEXP .Call(SEXP a, SEXP b, ...)
```

A good fit, it turns out (cont.)

- An **SEXP** (or S-Expression Pointer) is used for *everything*
- (An older C trick approximating object-oriented programming)
- We can ignore the details but retain that
 - everything in R is a **SEXP**
 - the **SEXP** is self-describing
 - can matrix, vector, list, function, ...
 - 27 types in total
- The key thing for Rcpp is that via C++ features we can map
 - each of the (limited number of) **SEXP** types
 - to a specific C++ class representing that type
 - and the conversion is automated back and forth

Other good reasons

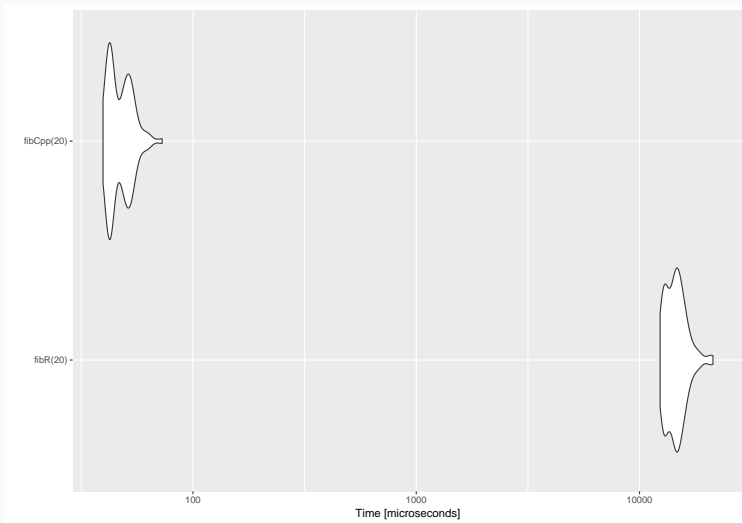
- It is *fast* – compiled C++ is hard to beat in other languages
 - (That said, you can *of course* write bad and slow code....)
- It is *very general* and widely used
 - *many libraries*
 - many tools
- It is fairly universal:
 - just about anything will have C interface so C++ can play
 - just about any platform / OS will have it

Key Features

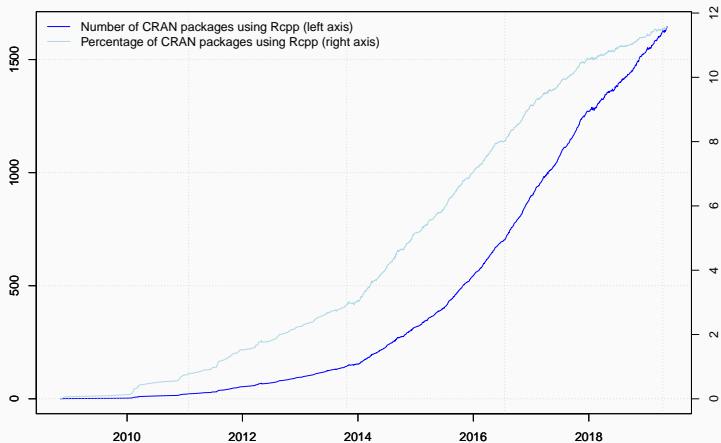
- *(Fairly) Easy to learn* as it really does not have to be that complicated – there are numerous examples
- *Easy to use* as it avoids build and OS system complexities thanks to the R infrastructure
- *Expressive* as it allows for *vectorised C++* using *Rcpp Sugar*
- *Seamless* access to all R objects: 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 directly or via eg *Rcpp modules*

RCPP SPEED ILLUSTRATION

Benchmark on Fibonacci(20) between C++ and R – note the log scale!



Growth of Rcpp usage on CRAN



Data current as of May 12, 2019.

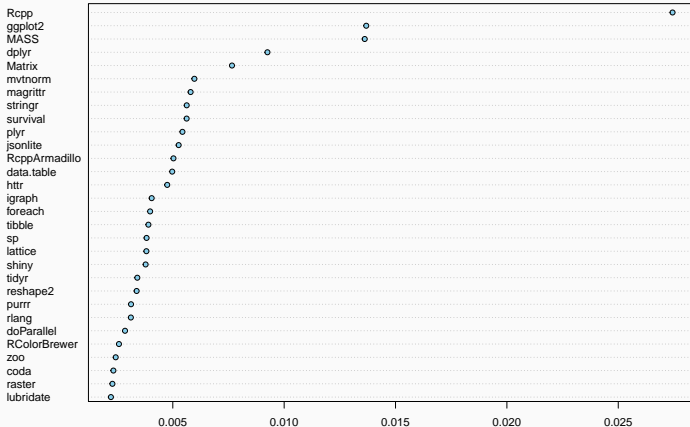
Rcpp is currently used by

- 1651 CRAN packages
- 176 BioConductor packages (with 38 added since last year)
- an unknown (but “large”) number of GitHub projects

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

```
##      Rcpp ggplot2      MASS      dplyr  Matrix  
##    2.744   1.369   1.362   0.925   0.766
```

Top 30 of Page Rank as of May 2019



PERCENTAGE OF COMPILED PACKAGES

```
db <- tools::CRAN_package_db() # added in R 3.4.0
## rows: number of pkgs, cols: different attributes
nTot <- nrow(db)
## all direct Rcpp reverse depends, ie packages using Rcpp
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 14266    1647      3579          46.01844
```

HOW: BRIEF RCPP INTRO

```
library(Rcpp)  
evalCpp("2 + 2")
```

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

This function can validate your installation.

It takes the supplied expression, wraps enough code around it to make a compilable function, compiles, links and loads it – to evaluate the C++ expression.

Here we skip all details about Rcpp installations. It just works *e.g.* on the (free) RStudio Cloud and in most normal system – see the documentation for more. As always, Windows may be hardest as you may have to install another R toolchain: **Rtools**.

FIRST STEPS: CPPFUNCTION()

```
library(Rcpp)
cppFunction("double fib(double n) { \
  if (n < 2) return(n); \
  return(fib(n-1) + fib(n-2)); \
}")
fib(30)
```

```
## [1] 832040
```

Creates R-callable function from a C++ function.

Finds function identifier in supplied string, creates R function of same name.

Useful for quick tests.

Can use additional headers and library (see documentation).

sourceCpp()

- 'sources' a file and compiles, links, loads
- file can contain multiple functions
- functions that are 'tagged' with `// [[Rcpp::export]]` become callable
- can contain non-exported helper functions
- use:

```
sourceCpp("someFile.cpp") # with or without path
```

FIRST STEPS: SOURCECPP()

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

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

/** R
timesTwo(42)
*/
```

This is a shortened (comments-removed) version of the file currently included when you say 'File -> New File -> C++' in RStudio.

FIRST STEPS: SOURCECPP()

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

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

/** R
timesTwo(42)
*/
```

Key features:

Rcpp header and
namespace

One exported
function `timesTwo()`

An automatically
executed (!!) R call for
tests and demos

Try it!

Quick Demo

```
Rcpp::sourceCpp("code/timestwo.cpp") # runs demo too
```

```
##
```

```
## > timesTwo(42)
```

```
## [1] 84
```

```
timesTwo(c(5,10,20)) # vectorized like R
```

```
## [1] 10 20 40
```

EXAMPLE: COLUMN SUMS

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector colSums(Rcpp::NumericMatrix mat) {
    size_t cols = mat.cols();
    Rcpp::NumericVector res(cols);
    for (size_t i=0; i<cols; i++) {
        res[i] = sum(mat.column(i));
    }
    return(res);
}
```

Key Elements

- `NumericMatrix` and `NumericVector` go-to types for matrix and vector operations on floating point variables
- We prefix with `Rcpp::` to make the namespace explicit
- Accessor functions `.rows()` and `.cols()` for dimensions
- Result vector allocated based on number of columns `column`
- Function `column(i)` extracts a column, gets a vector, and `sum()` operates on it
- That last `sum()` was internally vectorised, no need to loop over all elements

```
Rcpp::sourceCpp("code/colSums.cpp")
```

```
# test it
```

```
colSums(matrix(1:16, 4, 4))
```

```
## [1] 10 26 42 58
```

```
# base R for comparison
```

```
apply(matrix(1:16, 4, 4), 2, sum)
```

```
## [1] 10 26 42 58
```

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

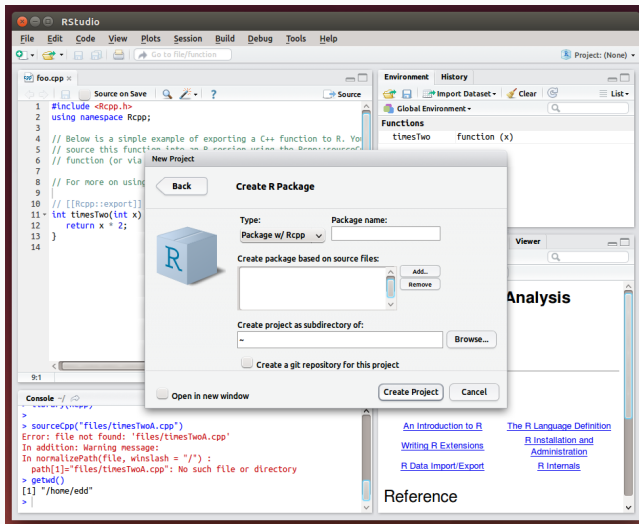
Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette [Rcpp-packages](#) has fuller details.

As of May 2019, there are 1651 CRAN and 176 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

PACKAGES AND RCPP

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



The screenshot shows the RStudio interface. In the background, a C++ source file named `foo.cpp` is open, containing the following code:

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
3
4 // Below is a simple example of exporting a C++ function to R. You
5 // source this function into an R session using the Rcpp::sourceCpp
6 // function (or via the R console).
7
8 // For more on using Rcpp, see the Rcpp website:
9 // http://www.Rcpp.org
10 // [[Rcpp::export]]
11 int timesTwo(int x) {
12   return x * 2;
13 }
14
```

The `Environment` pane shows a function named `timesTwo` of type `function (x)`.

The `Console` pane shows the following output:

```
> sourceCpp("files/timesTwoA.cpp")
Error: file not found: 'files/timesTwoA.cpp'
In addition: Warning message:
In normalizePath(file, winslash = "/") :
  path[1]="files/timesTwoA.cpp": No such file or directory
> getwd()
[1] "/home/edd"
```

The `Console` pane also displays a `Reference` section with the following links:

- [An Introduction to R](#)
- [Writing R Extensions](#)
- [R Data Import/Export](#)
- [The R Language Definition](#)
- [R Installation and Administration](#)
- [R Internals](#)

The `Create R Package` dialog box is open in the foreground. It features a `Back` button, a `Create R Package` title, and a `Package w/ Rcpp` dropdown menu. The `Package name:` field is empty. The `Create package based on source files:` section contains an empty list box with `Add...` and `Remove` buttons. The `Create project as subdirectory of:` field is empty, with a `Browse...` button. There is an unchecked checkbox for `Create a git repository for this project` and an unchecked checkbox for `Open in new window`. The `Create Project` and `Cancel` buttons are at the bottom right.

`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 (or three) ways to link to external libraries

- *Full copies*: Do what several packages (e.g. RcppMLPACK (v1), RVowpalWabbit) do and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries*: Do what RcppGSL or RcppMLPACK (v2) do 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.

RELATED APPROACHES

R Interface to Python

The **reticulate** package provides a comprehensive set of tools for interoperability between Python and R. The package includes facilities for:

- Translation between R and Python objects (for example, between R and Pandas data frames, or between R matrices and NumPy arrays).
- Calling Python from R in a variety of ways including R Markdown, sourcing Python scripts, importing Python modules, and using Python interactively within an R session.
- Flexible binding to different versions of Python including virtual environments and Conda environments.



Reticulate embeds a Python session within your R session, enabling seamless, high-performance interoperability. If you are an R developer that uses Python for some of your work or a member of data science team that uses both languages, reticulate can dramatically streamline your workflow!

Source: <https://rstudio.github.io/reticulate/>

reticulate

- Written to support **tensorflow** and **keras**
- Already used by several packages including
 - **greta**: think stan or bugs, but on tensorflow
 - **spacyr**: accesses the [spaCy](#) NLP engine
 - **h2o4gpu**: access to [h2o.ai](#) GPU-based ML solvers
- Also used by **XRPython**
- Uses Rcpp

The `RcppCNPY` package lets us load and save NumPy files (by wrapping the C library `cnpy`).

```
library(RcppCNPY)
mat <- npyLoad("fmat.npy")
vec <- npyLoad("fvec.npy")

mat2 <- npyLoad("fmat.npy.gz")
```

But `reticulate` lets us load and save NumPy files directly!

```
library(reticulate)
np <- import("numpy")
mat <- np$load("fmat.npy")
vec <- np$load("fvec.npy")

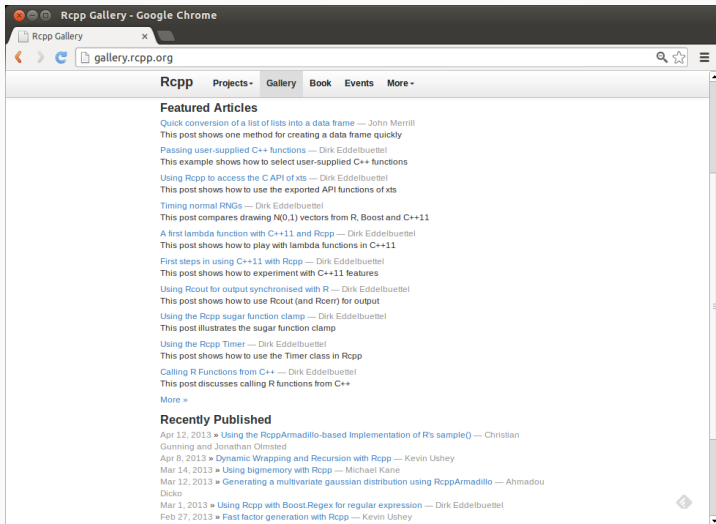
## compressed data: import gzip
gz <- import("gzip")
## use it to create handle to uncompressed file
mat2 <- np$load(gz$GzipFile("fmat.npy.gz", "r"))
```

See the vignettes in the `RcppCNPY` package for more.

MORE

Documentation and Examples

- The package comes with nine 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 list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has a fair number of posts too.
- 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

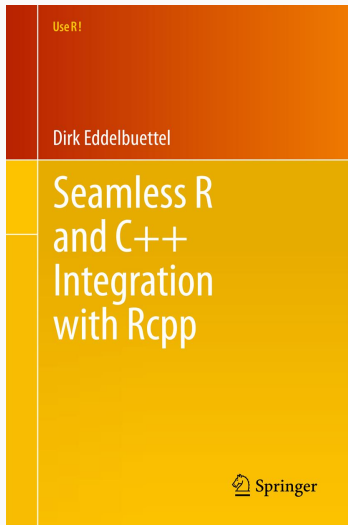
Apr 8, 2013 » [Dynamic Wrapping and Recursion with Rcpp](#) — Kevin Ushey

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 <http://dirk.eddelbuettel.com/presentations/>

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

mail dirk@eddelbuettel.com

github [@eddelbuettel](#)

twitter [@eddelbuettel](#)