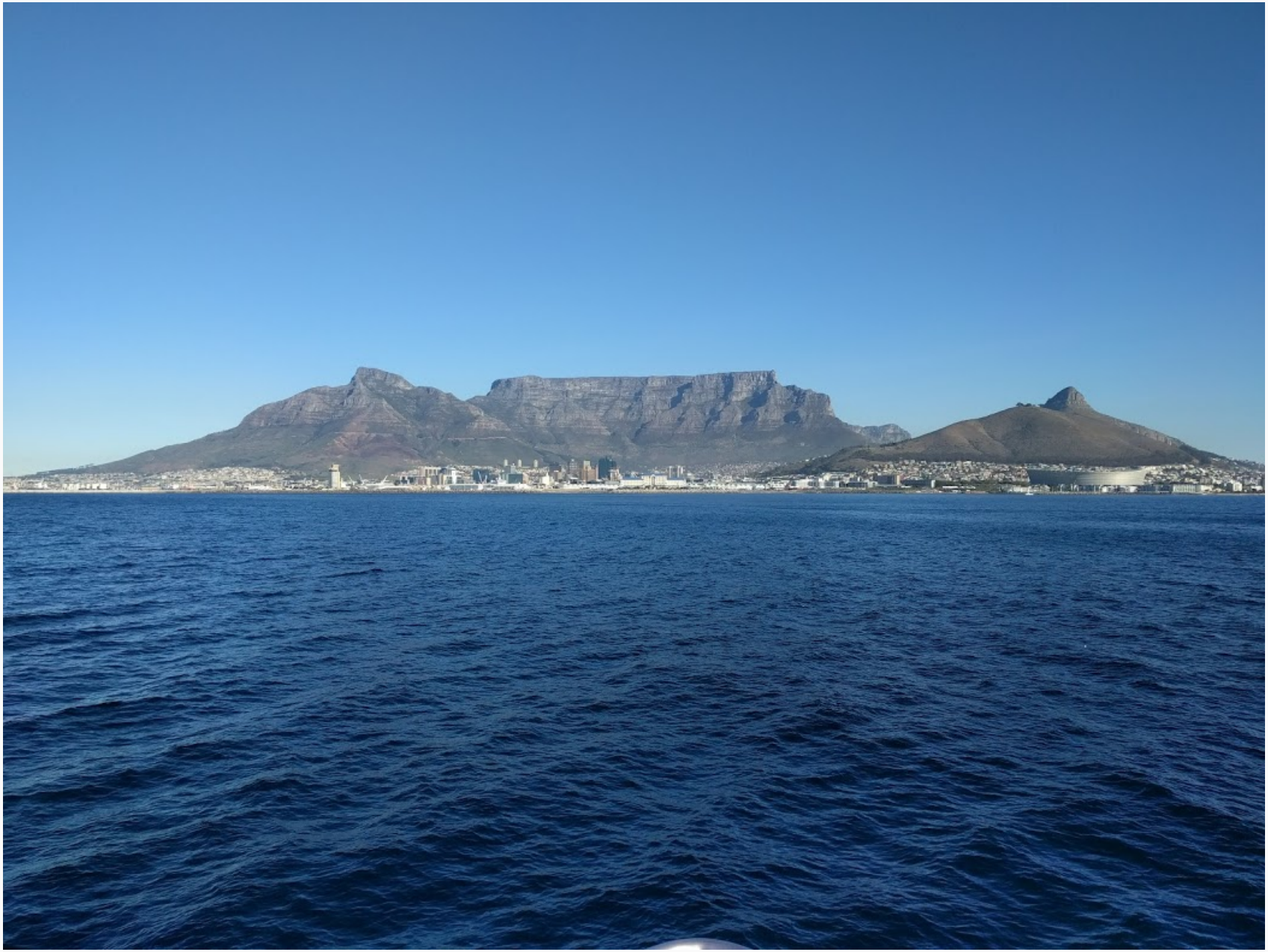
A Gentle Introduction to Rcpp

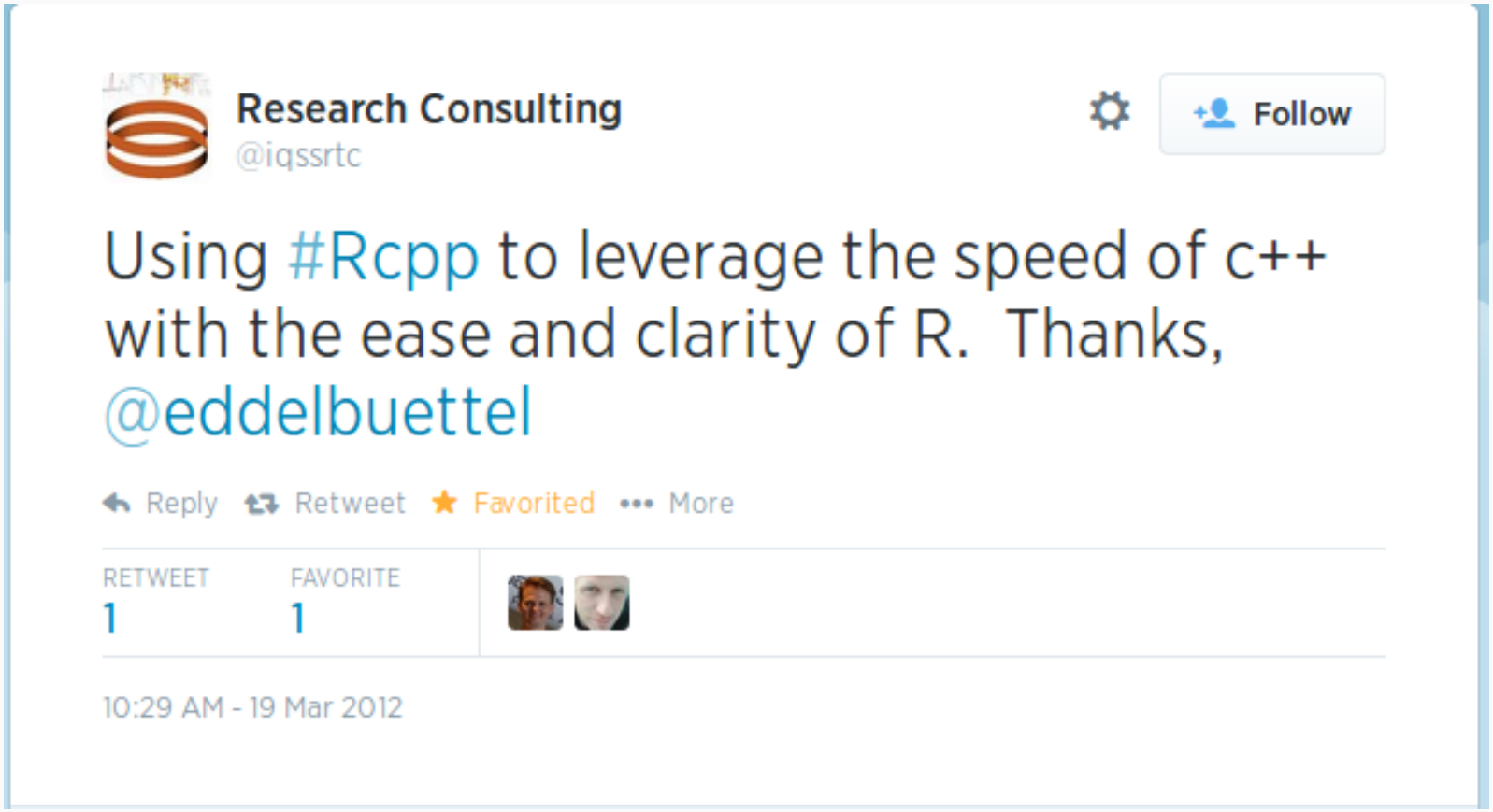
Dirk Eddelbuettel

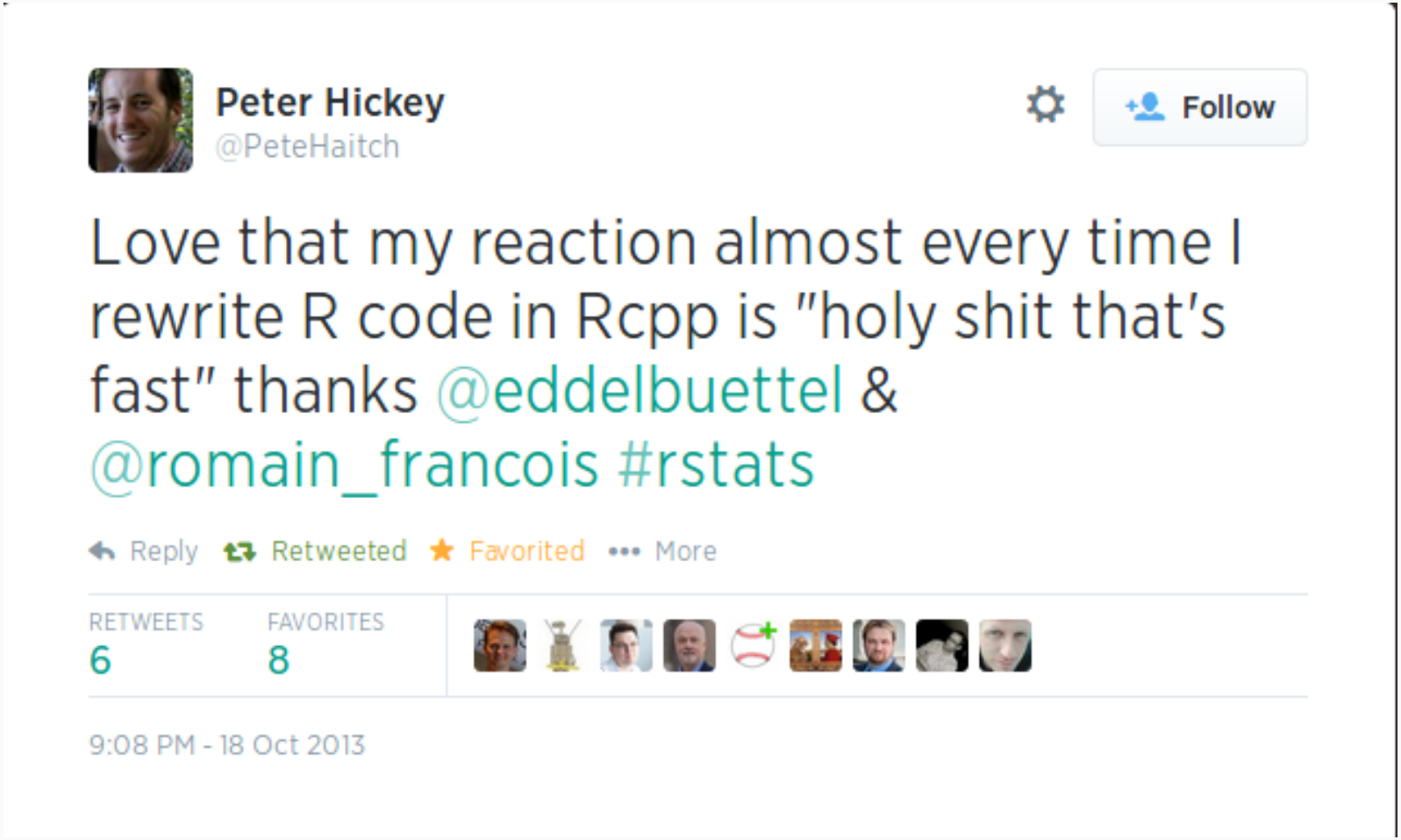
Presentation to the *Cape Town R User Group* on 9 July 2016

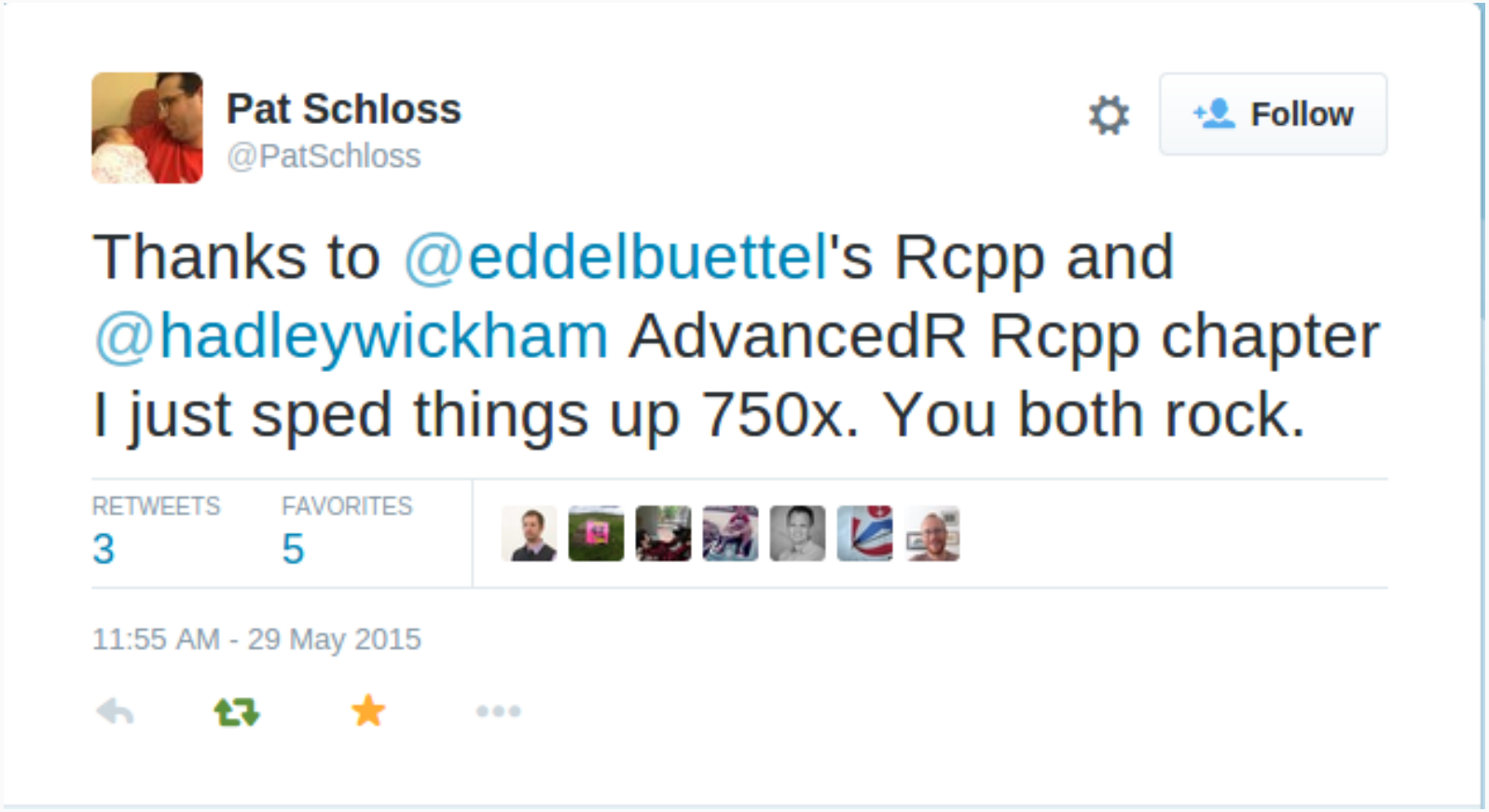
Debian and R Projects



Introduction via Twitter


Research Consulting
@iqssrtc
Follow
Using #Rcpp to leverage the speed of c++ with the ease and clarity of R. Thanks, @eddelbuettel
Reply
Retweet
Favorited
More
RETWEET
1
FAVORITE
1
10:29 AM - 19 Mar 2012


Peter Hickey
@PeteHaitch
Follow
Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois #rstats
Reply
Retweeted
Favorited
More
RETWEETS
6
FAVORITES
8
9:08 PM - 18 Oct 2013

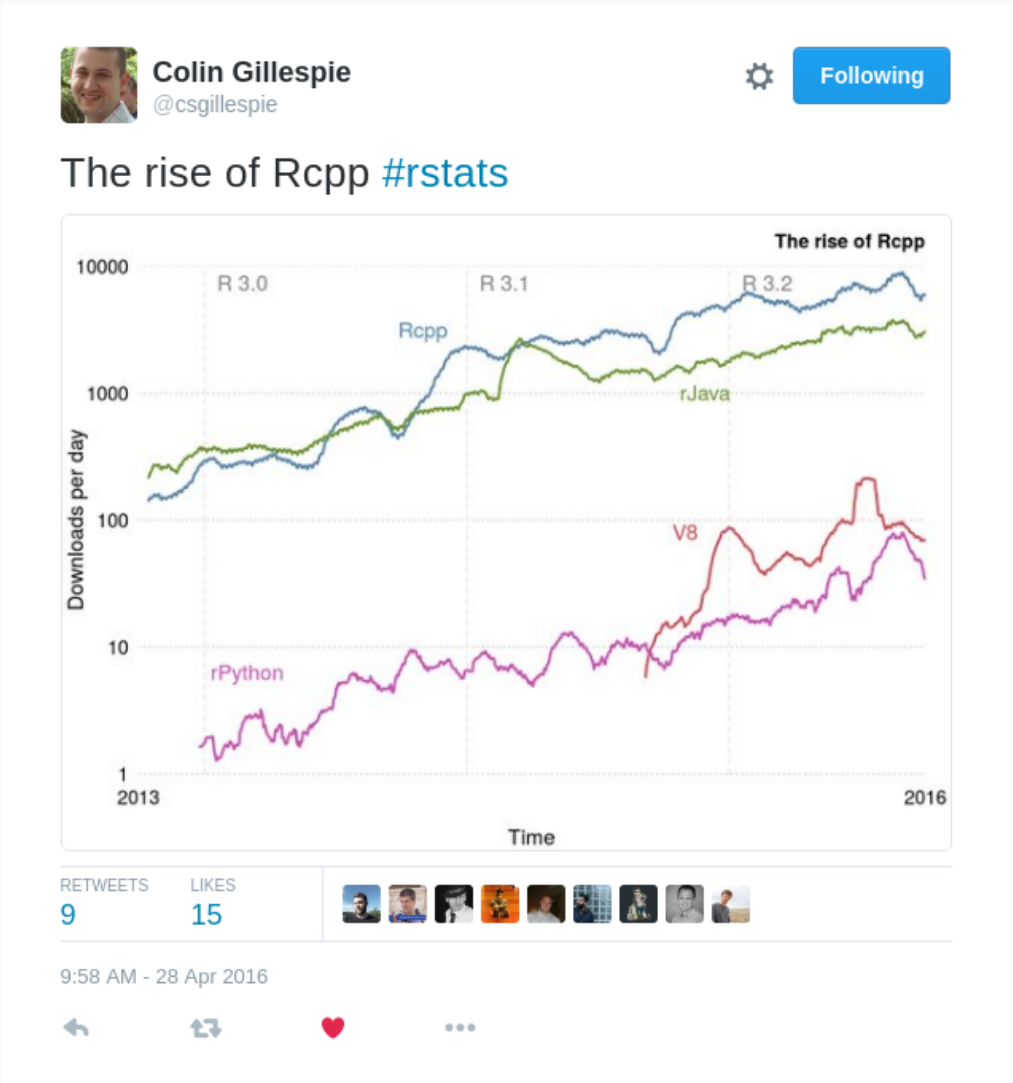

Pat Schloss
@PatSchloss
Follow
Thanks to @eddelbuettel's Rcpp and @hadleywickham AdvancedR Rcpp chapter I just sped things up 750x. You both rock.
RETWEETS
3
FAVORITES
5
11:55 AM - 29 May 2015


Rich FitzJohn
@rgfitzjohn
Follow
Writing some code using #rstats plain C API and realising/remembering quite how much work Rcpp saves - thanks @eddelbuettel
RETWEETS
5
FAVORITES
8
5:45 PM - 6 Mar 2015


Romain François
@romain_francois
Following
"Rcpp is one of the 3 things that changed how I write #rstats code". @hadleywickham at #EARL2014
RETWEETS
3
FAVORITES
7
3:19 AM - 16 Sep 2014

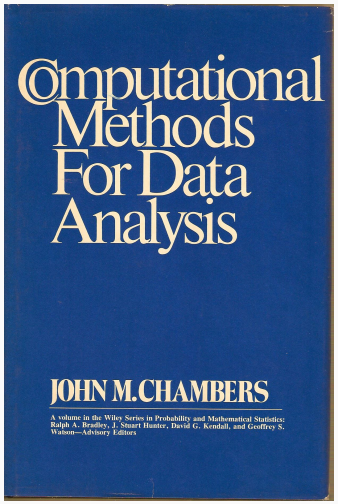

Karl Broman
@kwbroman
Following
@eddelbuettel @romain_francois Have I emphasized how much I #Rcpp?
LIKES
8
9:12 PM - 27 May 2016


boB Rudis
@hrbrmstr
Follow
Gosh, Rcpp is the bee's knees (cc: @eddelbuettel) #rstats
LIKES
6
9:08 AM - 18 Feb 2016

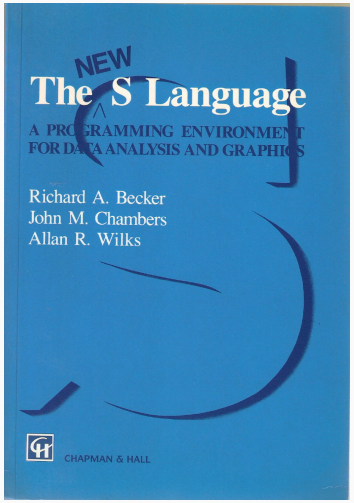

Colin Gillespie
@csgillespie
Following
The rise of Rcpp #rstats
The rise of Rcpp
10000
1000
100
10
1
Downloads per day
R 3.0
R 3.1
R 3.2
Rcpp
rJava
V8
rPython
2013
2016
Time
RETWEETS
9
LIKES
15
9:58 AM - 28 Apr 2016

Extending R

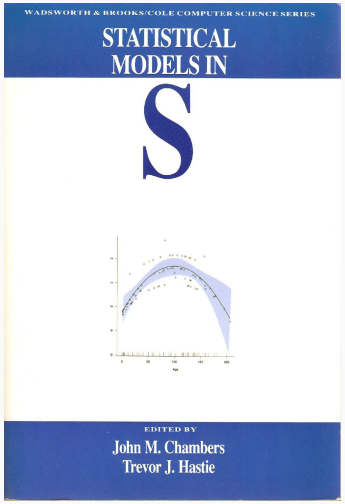
Why R? : *Programming with Data*



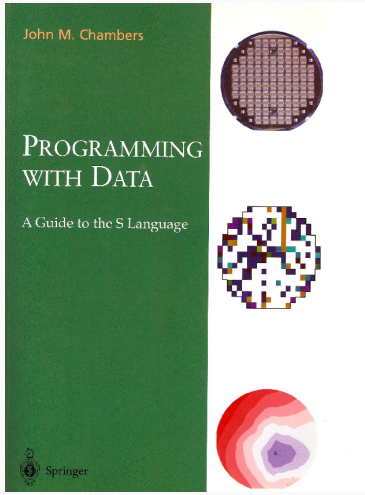
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.



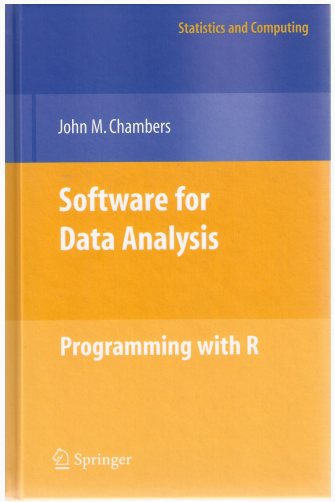
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



Chambers and Hastie. *Statistical Models in S*. Chapman & Hall, 1992.



Chambers. *Programming with Data*. Springer, 1998.



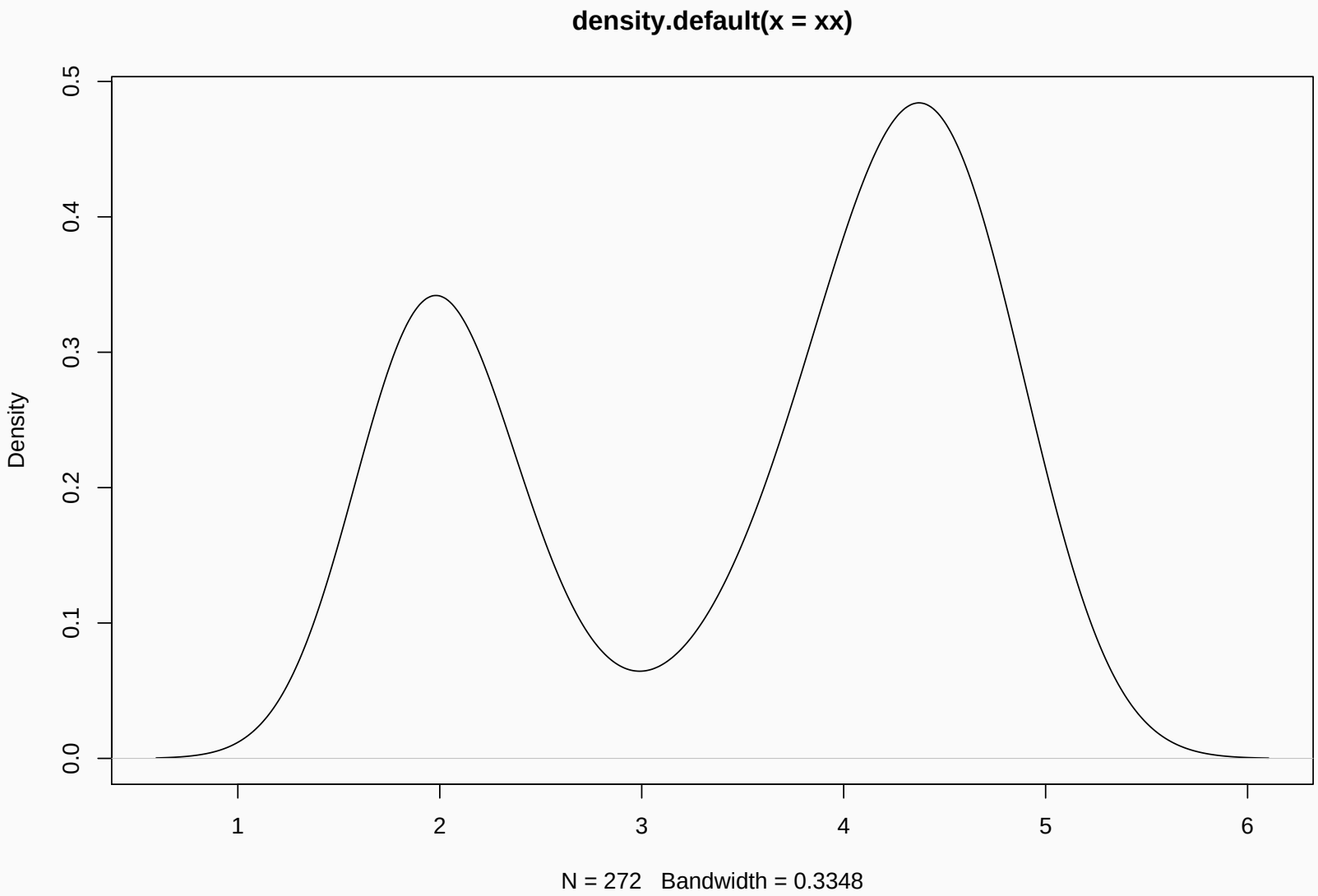
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

Thanks to John Chambers for sending me high-resolution scans of the covers of his books.

A Simple Example

```
xx <- faithful[,"eruptions"]
fit <- density(xx)
plot(fit)
```

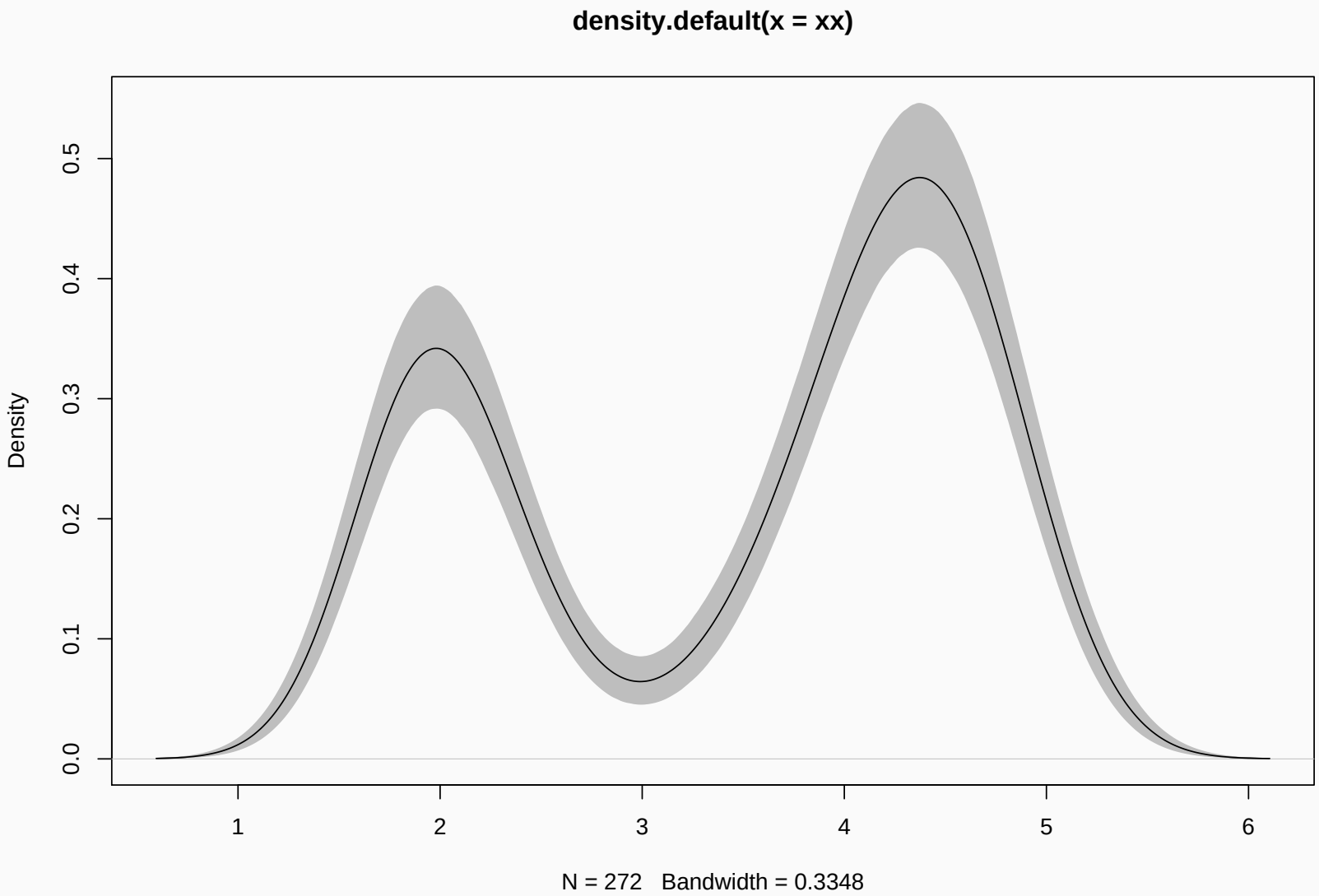
A Simple Example



A Simple Example - Refined

```
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)
```

A Simple Example - Refined



So Why R?

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.

So Why R?

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*

Why Extend R?

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.*

Why Extend R?

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.*

Why Extend R?

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

Why Extend R?

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)

Why C++?

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.

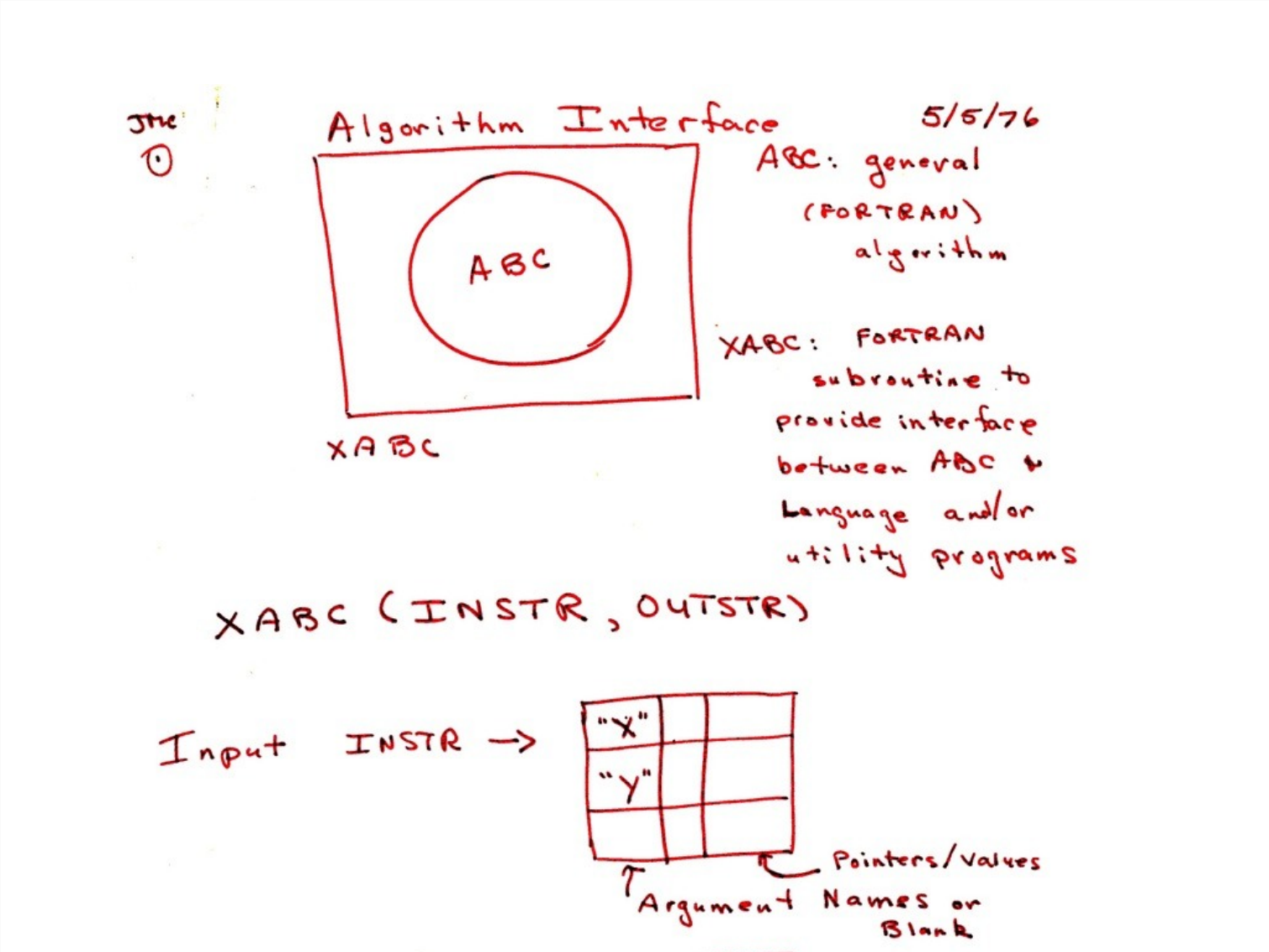
NB: Meyers original list of four languages appeared years before C++11.

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 is a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded

Interface Vision

Bell Labs, May 1976



Interface Vision

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
  - An accessible 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 infrastrucure
- 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

Speed Example (due to StackOverflow)

Consider a function defined as

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

Speed Example in R

R implementation and use:

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

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

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

Speed Example Timed

Timing:

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

```
##    test replications elapsed relative
## 1 f(10)          100   0.020     1.00
## 2 f(15)          100   0.227    11.35
## 3 f(20)          100   2.466   123.30
```

Speed Example in C / C++

A C or C++ solution can be equally simple

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

But how do we call it from R?

Speed Example in C / C++

But Rcpp makes this *much* easier:

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

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

Speed Example Comparing R and C++

Timing:

```
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   0.063    1.000
## 1 f(25)          100  28.877  458.365
```

A nice gain of a few orders of magnitude.

Another Angle on Speed

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.

Basic Usage

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] 1.797693e+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
```

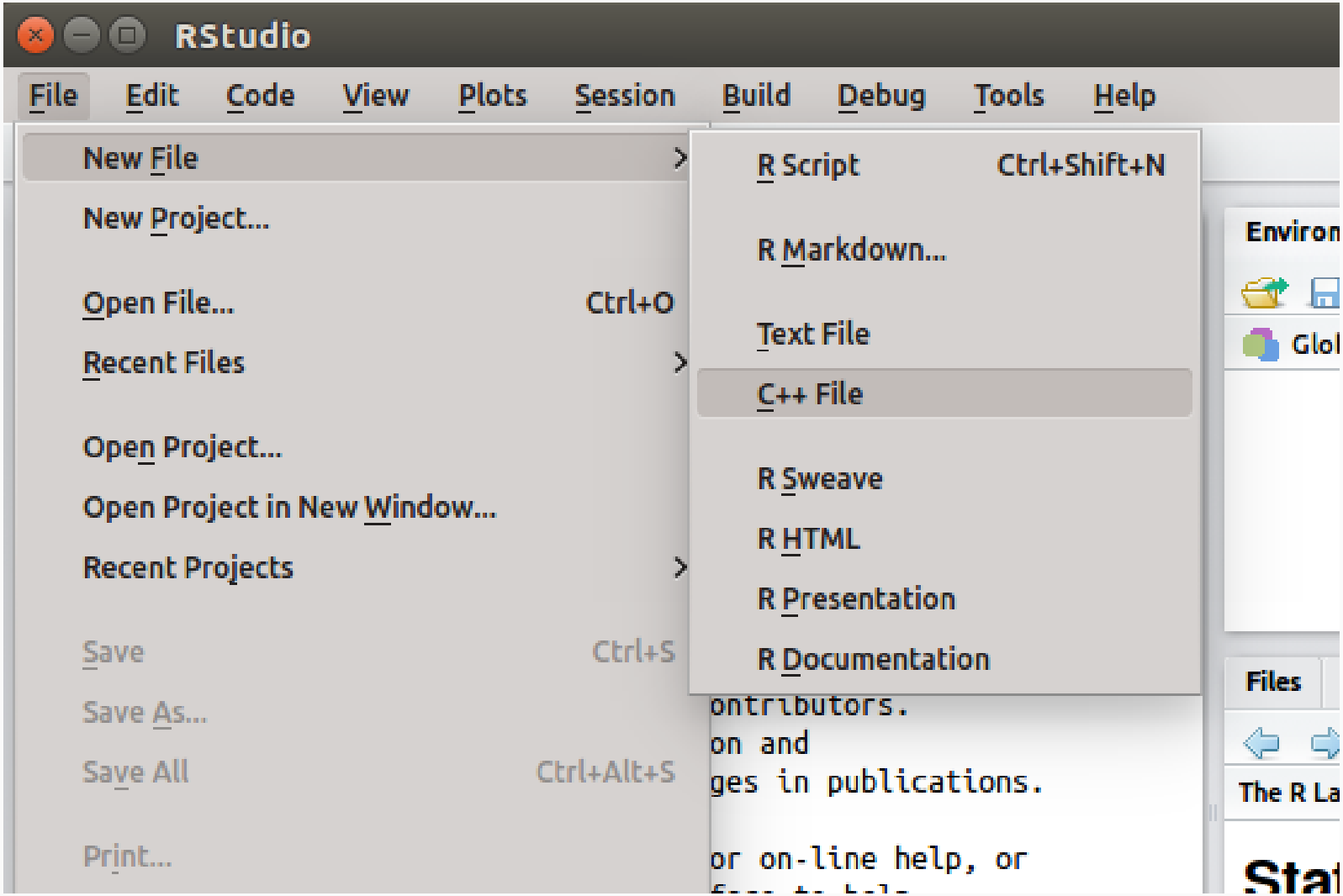
Basic Usage: sourceCpp()

`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).

Basic Uage: RStudio



Basic Uage: RStudio (Cont'ed)

The following file gets created:

```
#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

Basic Usage: Packages

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

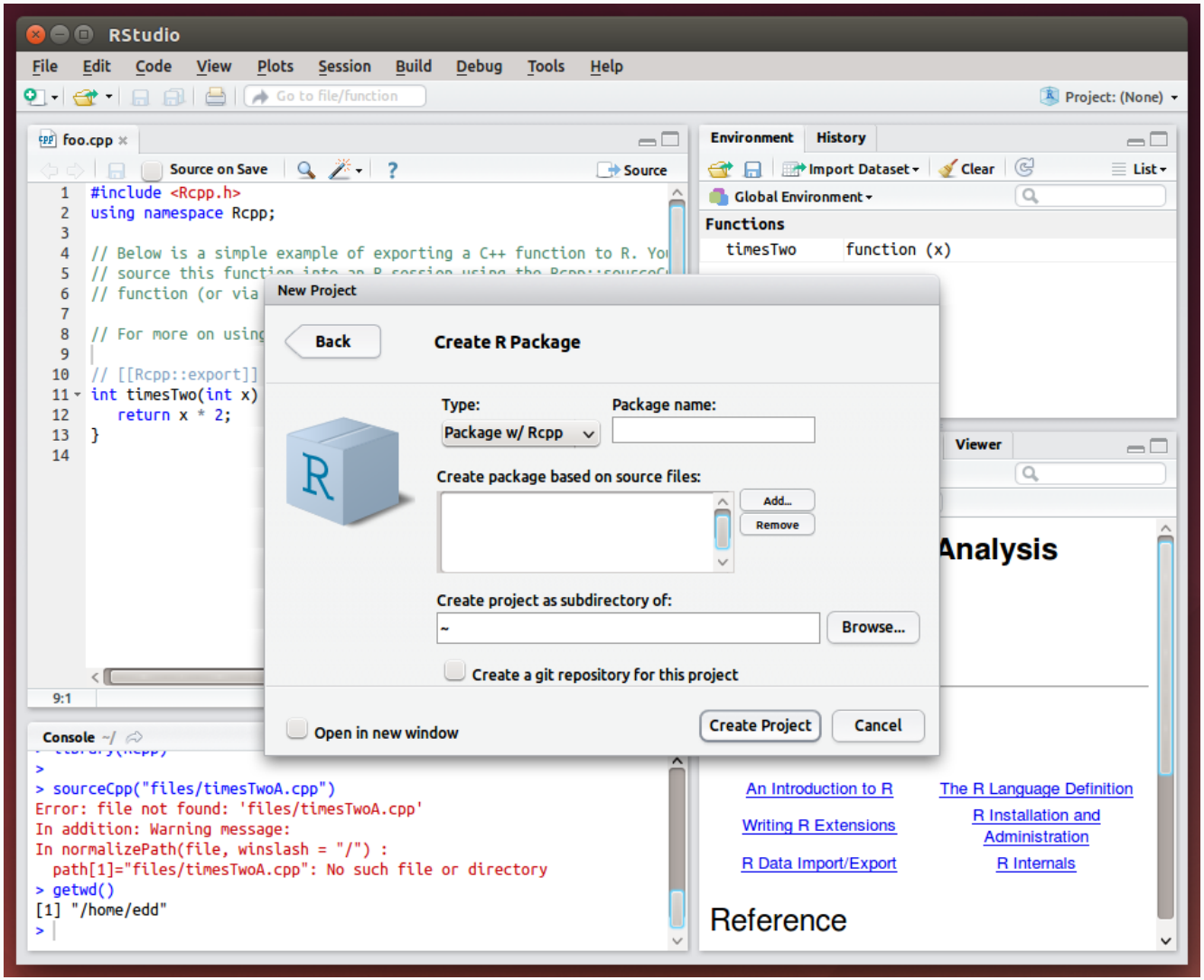
Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette Rcpp-packages has fuller details.

As of early June 2016, there are 680 packages on CRAN which use Rcpp, and a further 72 on BioConductor — with working, tested, and reviewed examples.

Packages and Rcpp

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



Packages and Rcpp

`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 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] 3.156000 3.155200 3.139000 3.141008
```

Syntactic 'sugar': Simulating π in C++

The neat thing about Rcpp sugar enables us to write C++ code that looks almost as compact.

```
#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 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
```

Syntactic 'sugar': Simulating π

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  15.357    1.657
## 2 piSugar(1e+06)          100   9.269    1.000
```

Syntactic 'sugar': Simulating π

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

Cumulative Sum: vector-cumulative-sum

A basic looped version:

```
#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;
}
```

Cumulative Sum: vector-cumulative-sum

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;
}
```

Cumulative Sum: vector-cumulative-sum

Or just Rcpp sugar:

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

Of course, all results are the same.

R Function Call from C++: r-function-from-c++

```
#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

```
// [[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));
}
```

Using Boost via BH: using-boost-with-bh

```
#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.

Vector Subsetting: subsetting

```cpp
#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];
}
```

Armadillo Eigenvalues: armadillo-eigenvalues

```
#include <RcppArmadillo.h>

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

// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
    return arma::eig_sym(M);
}
```

Armadillo Eigenvalues: armadillo-eigenvalues

```
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.
```

Create xts from in C++: creating-xts-from-c++

```
#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;
}
```

RcppMLPACK: K-Means Example

```
#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

test	replications	elapsed	relative	user.self	sys.self
mlKmeans(t(wine), 3)	100	0.028	1.000	0.028	0.000
kmeans(wine, 3)	100	0.947	33.821	0.484	0.424

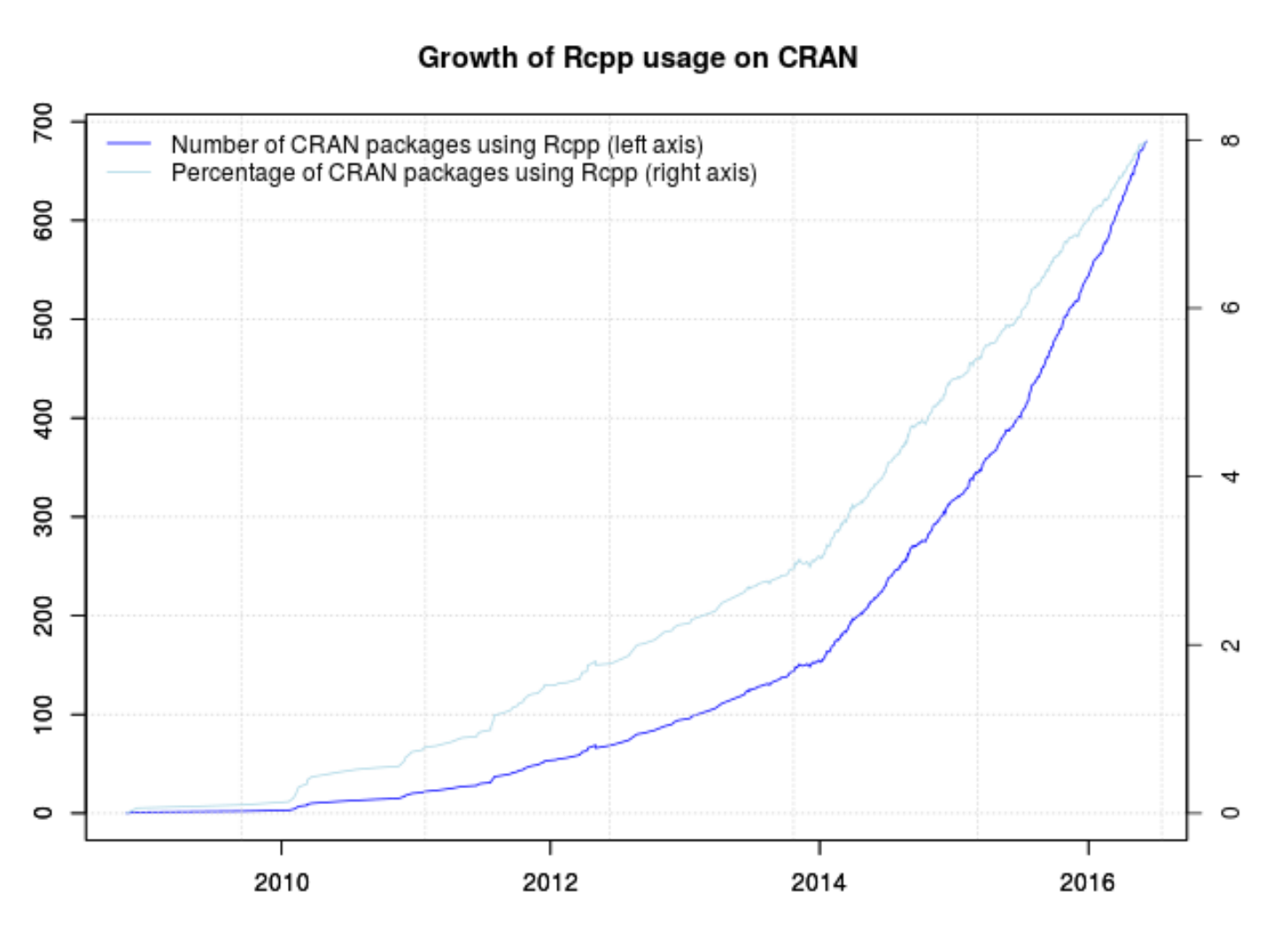
Table taken 'as is' from RcppMLPACK vignette.

RcppMLPACK: Nearest Neighbors Example

```
#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(&tree, data, true);

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

    return List::create(Named("clusters") = coverTreeNeighbors,
                        Named("result")   = coverTreeDistances);

}
```

More

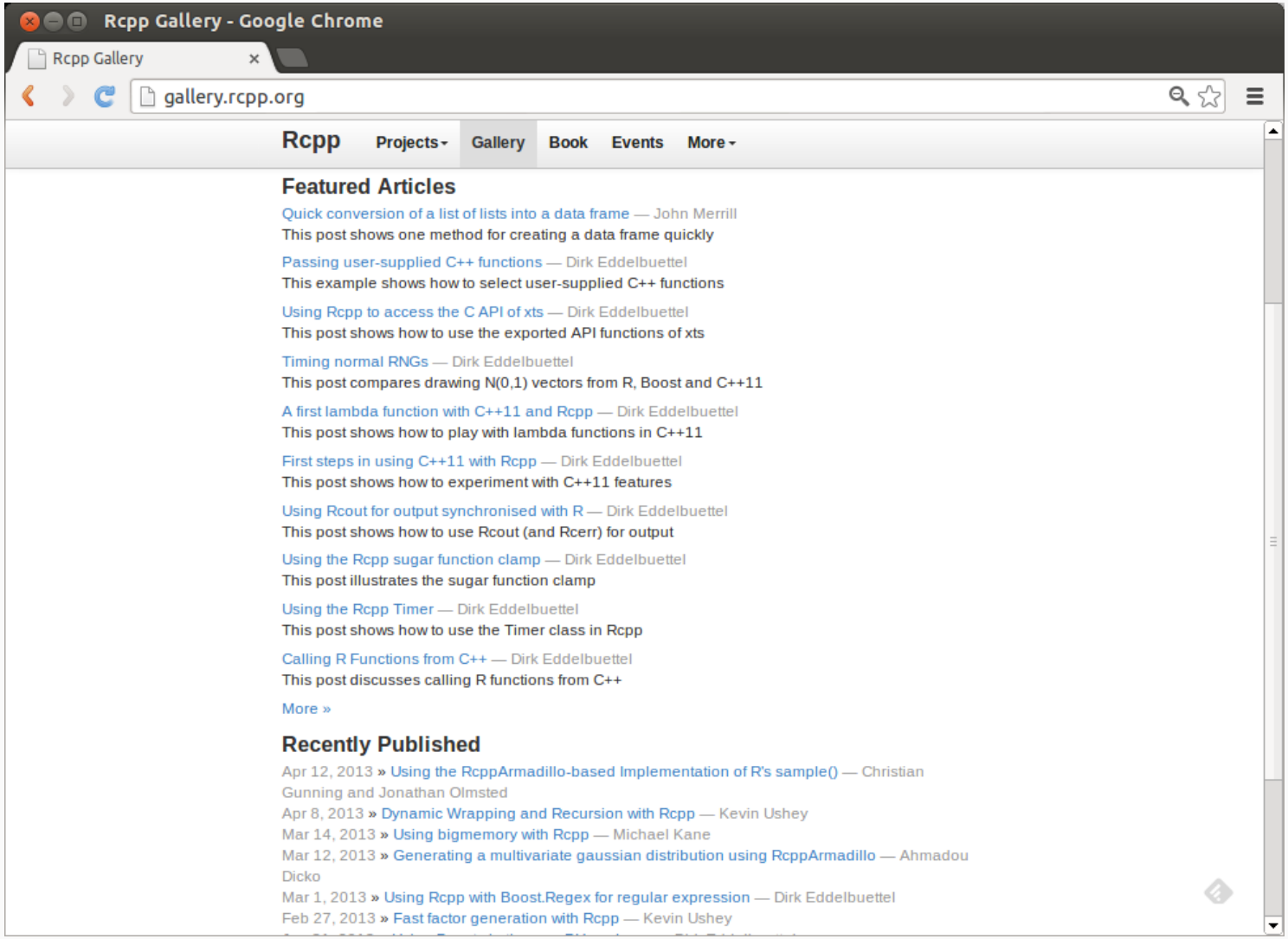
Used by about 680 CRAN Packages



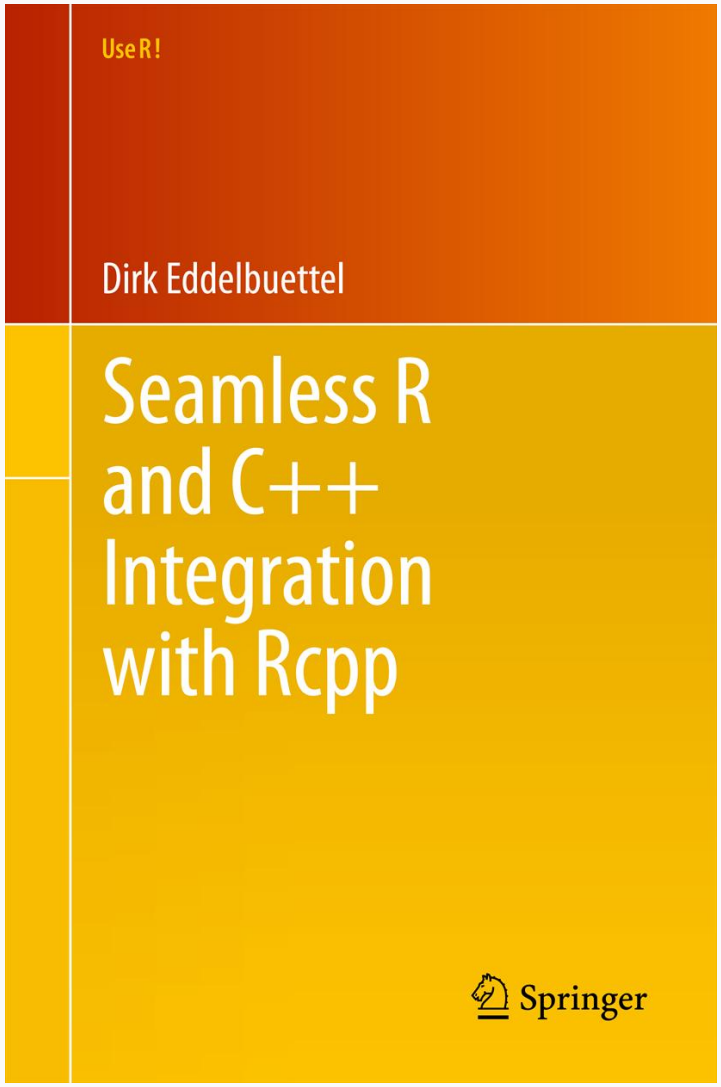
Documentation

- 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 fair number of posts too.
- And a number of blog posts introduce/discuss features.

Rcpp Gallery



The Rcpp book



On sale since June 2013.

Questions?

Contact

http://dirk.eddelbuettel.com

dirk@eddelbuettel.com

@eddelbuettel