R and 'Faster Data'

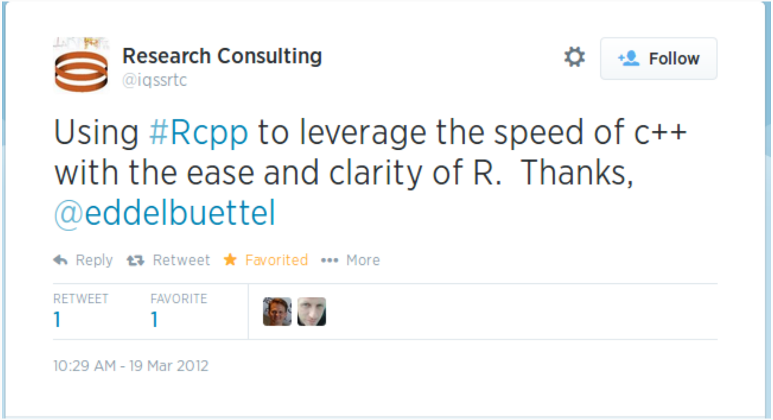
The Case for Rcpp

Dirk Eddelbuettel

ISM HPCCON 2015 & ISM HPC on R Workshop
The Institute of Statistical Mathematics, Tokyo, Japan
October 9 - 12, 2015

Introduction

A Very Kind Tweet



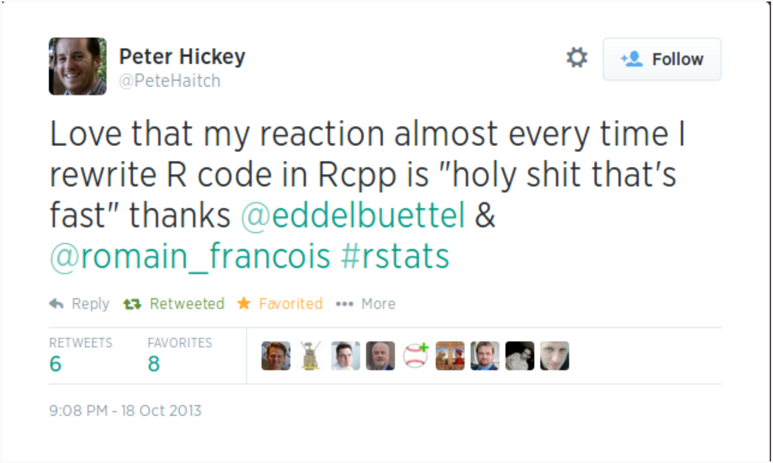
Research Consulting
@iqssrtc

Using #Rcpp to leverage the speed of c++ with the ease and clarity of R. Thanks, @eddelbuettel

RETWEET 1
FAVORITE 1

10:29 AM - 19 Mar 2012

And Another Tweet



Peter Hickey
@PeteHaitch

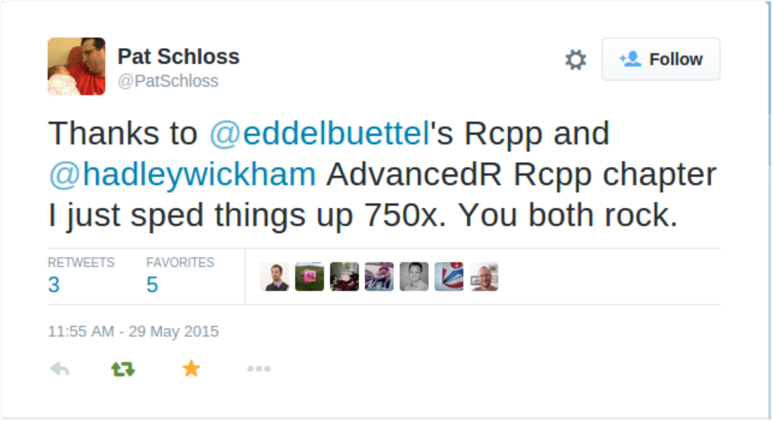
Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois #rstats

RETWEETS 6

FAVORITES 8

9:08 PM - 18 Oct 2013

And Yet Another Tweet



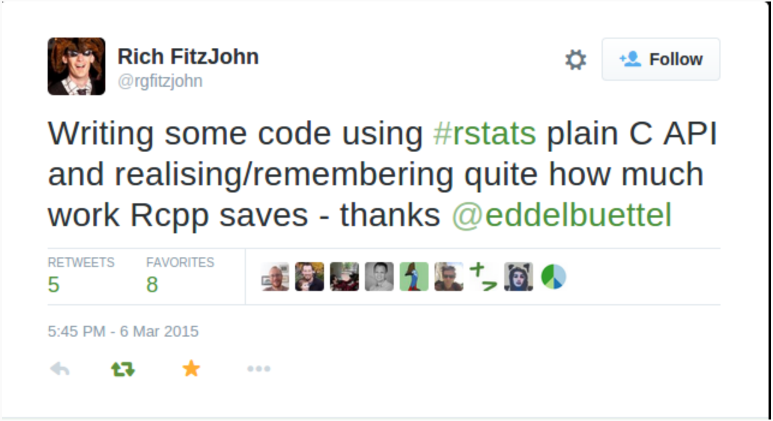
Pat Schloss
@PatSchloss

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

And Why Not Another Tweet



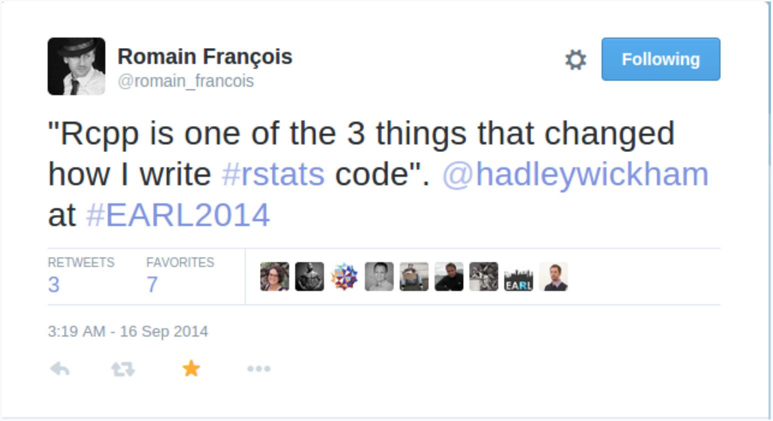
Rich FitzJohn
@rgfitzjohn

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

And Last But Not Least



Romain François
@romain_francois

"Rcpp is one of the 3 things that changed how I write #rstats code". @hadleywickham at #EARL2014

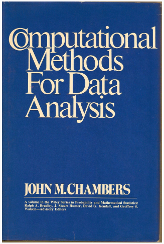
RETWEETS 3

FAVORITES 7

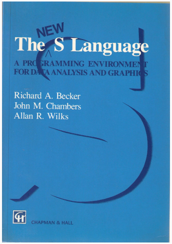
3:19 AM - 16 Sep 2014

Extending R

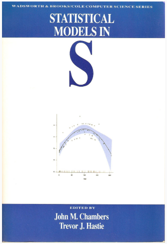
Why R? : *Programming with Data*


Computational
Methods
For Data
Analysis
JOHN M. CHAMBERS
A volume in the Wiley Series in Probability and Mathematical Statistics: Ralph A. Bradley, J. Stuart Hunter, David G. Kendall, and Geoffrey S. Watson—Advisory Editors

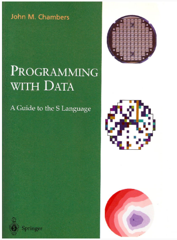
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.



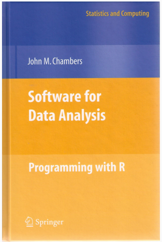
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



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



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



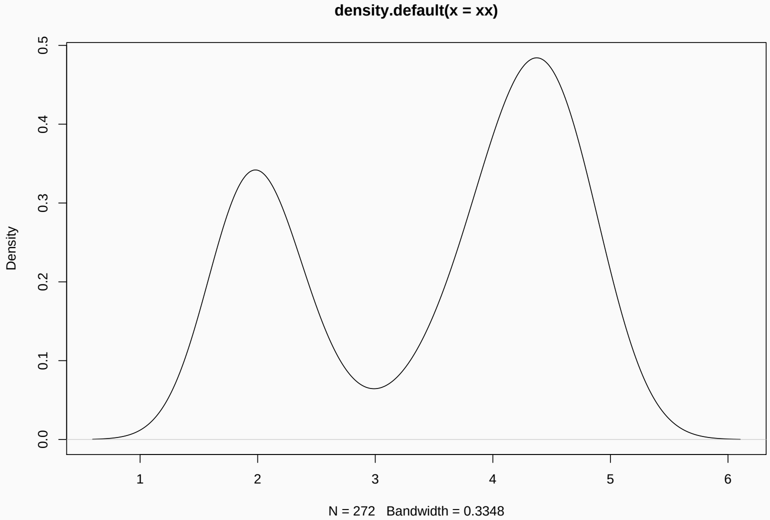
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

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

A Simple Example

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

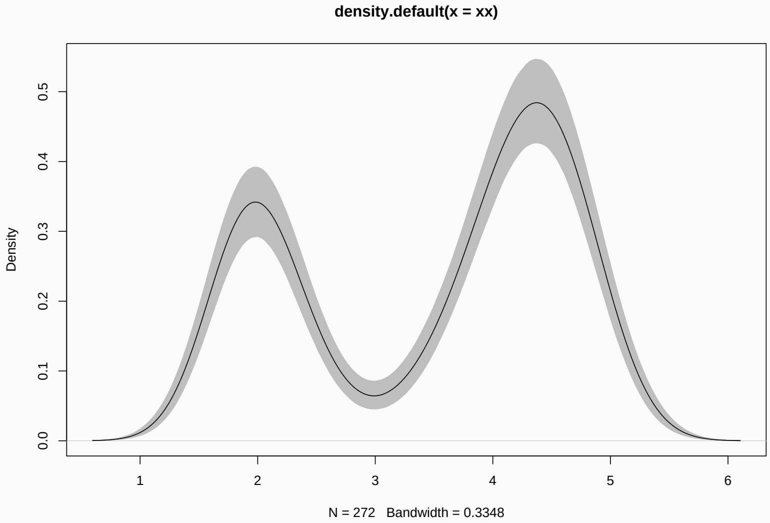
A Simple Example



A Simple Example - Refined

```r
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 about ~ 50 million 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.

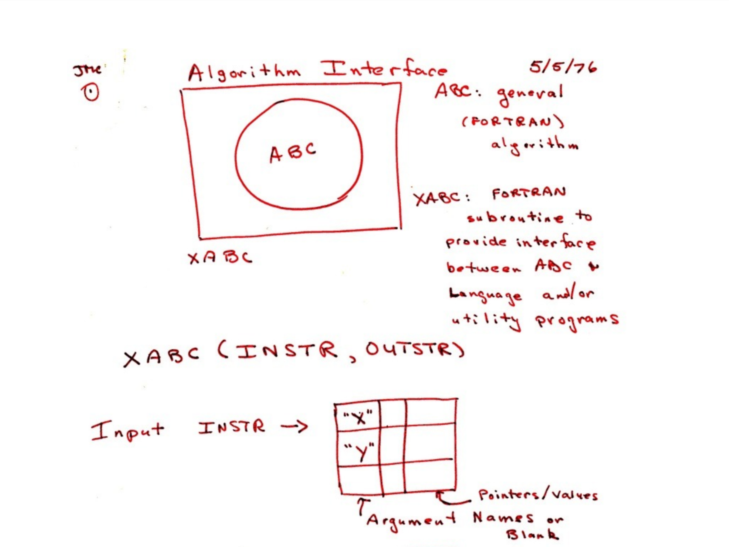
NB: Meyers original list of four languages appeared years before C++11.

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



JMC
⊙

Algorithm Interface 5/5/76

ABC: general (FORTRAN) algorithm

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

XABC (INSTR, OUTSTR)

Input INSTR →

"X"		
"Y"		

Argument Names or Blank

Pointers/Values

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

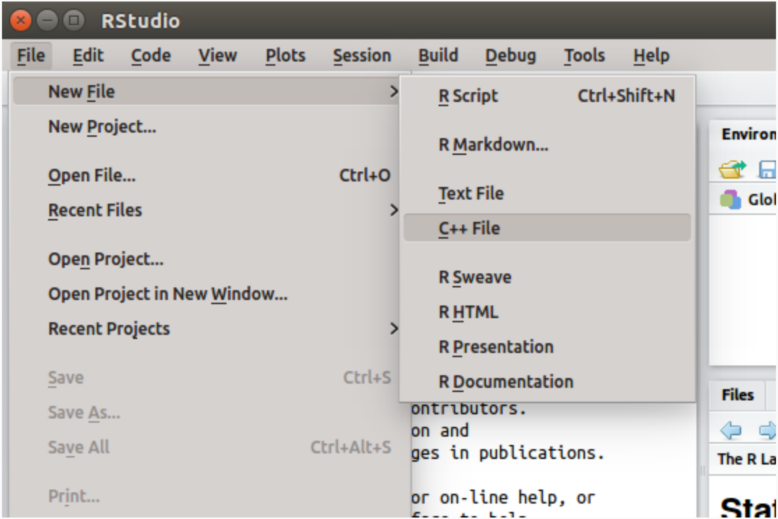
Why Rcpp?

- 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*

Getting Started

sourceCpp: Jumping Right In

RStudio makes starting very easy:



ontributors.
on and
ges in publications.

or on-line help, or

A First Example: 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)
*/
```

A First Example: Cont'ed

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

A First Example: Cont'ed

Try it:

- Save the file as, say, `timesTwo.cpp`
- You could a temporary directory, or a projects directory, or your desktop (keep it simple)
- Either press the *Source:* button or call `sourceCpp("thefile.cpp")` to compile it
- Then at the R prompt:

```r
## simple
timesTwo(21)
## more interesting
timesTwo(c(1,2,3,44,101))
```

cppFunction

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

```r
cppFunction("int times2(int x) { return 2*x; }")
times2(21)  # same identifier as C++ function
```

evalCpp

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
```

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

```
## [1] 1.797693e+308
```

Speed

Speed Example (due to StackOverflow)

Consider a function defined as

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

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:

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

```
##     test replications elapsed relative
## 1 f(10)          100   0.023    1.000
## 2 f(15)          100   0.542   23.565
## 3 f(20)          100   6.172  268.348
```

Speed Example in C / C++

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?

Matt's Example from useR! 2015

```
#include <R.h>
#include <Rinternals.h>

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

SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*
## need to compile, link, load, ...
fibonacci <- function(n) .Call("fibonacci_c", n)
sapply(0:10, fibonacci)
*/
```

One Minor Modification to Matt's Example

```c
#include <R.h>
#include <Rinternals.h>

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

// [[Rcpp::export]]
SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*** R
sapply(0:10, fibonacci_c)
*/
```

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    0.20      1.0
## 1 f(25)          100   66.22    331.1
```

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.

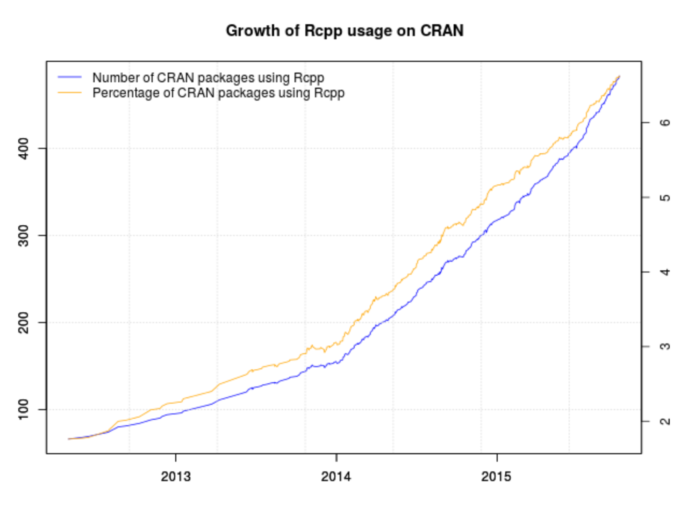
A good environment helps. RStudio integrates R and C++ development quite nicely (eg the compiler error message parsing is very helpful) and also helps with package building.

Speed Example Footnote Also Due to Matt

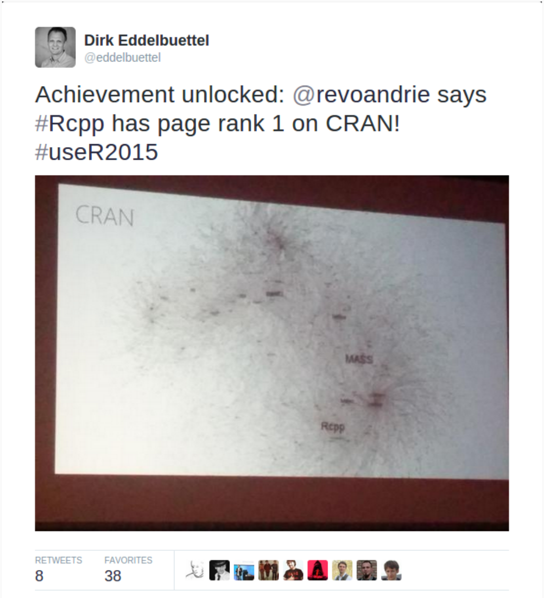
```cpp
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

constexpr int fibonacci_recursive_constexpr(const int n) {
    return n < 2 ? n : (fibonacci_recursive_constexpr(n - 1) +
                        fibonacci_recursive_constexpr(n - 2));
}

// [[Rcpp::export]]
int constexprFib() {
    const int N = 42;
    constexpr int result = fibonacci_recursive_constexpr(N);
    return result;
}
```

POPULARITY

USED BY 483 CRAN PACKAGES AS OF THIS WEEK



Page Rank One (According to Andrie de Vries)



Dirk Eddelbuettel
@eddelbuettel

Achievement unlocked: @revoandrie says #Rcpp has page rank 1 on CRAN! #useR2015

RETWEETS 8

FAVORITES 38

Case Study

Time Series Dashboard

Previous Status

- We have a lot of data circulating at work
- Market prices, positions, risk estimates, profit/loss, ...
- The used to be displayed in a one-off ‘display grid’
- But no history, and no plots

Time Series Dashboard

Easy R Fix

- Use Redis to cache data
- Redis is simple, well-established, widely used
- Excellent R package rredis by Bryan Lewis
- Use Shiny to access Redis and create ‘dashboards’
- We need to be fast enough to keep users engaged
- Goal is $\sim$ 250 msec (in-line with web UI research)

Time Series Dashboard

What does Redis do?

- Essentially a very fast and lightweight key/value store:
    - After `SET key value`
    - Do `GET key` to retrieve value
- APIs for multiple languages: C/C++, Python, Java, ...
- Can also store lists, sets, ...
- Can be coaxed to provide simple columnar data store
- Basic access: store strings, retrieve strings

Time Series Dashboard

What is wrong with that?

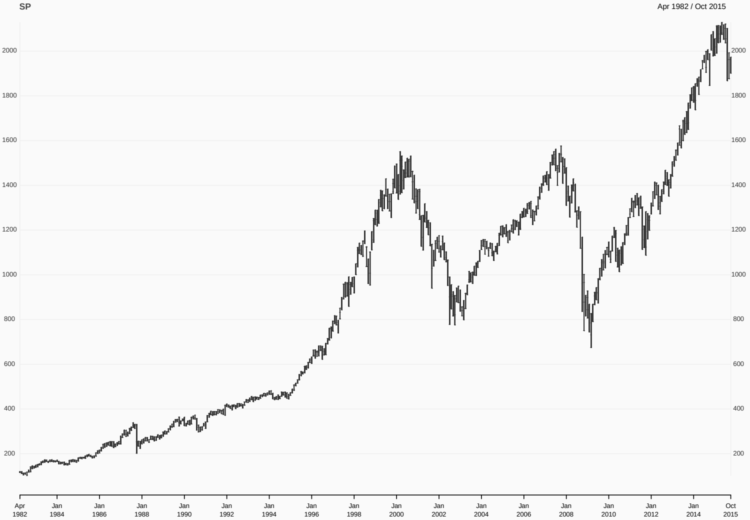
- String conversion ‘expensive’ when done repeatedly for a few thousand points
- Do string conversion in compiled code – RcppRedis
- A step better: R serialization and deserialization using RApiSerialize

Time Series Dashboard

Getting Data

```r
library(Quandl)
Quandl.api_key(yourAPIkeyhere)  # register, obtain key; anon possible too
sp <- Quandl("CHRIS/CME_SP1" , type="xts")
saveRDS(sp, file="data/quandl-sp1.rds")      # longer series
es <- Quandl("CHRIS/CME_ES1" , type="xts")
saveRDS(sp, file="data/quandl-es1.rds")      # more active
head(sp, 3)
```

Time Series Dashboard: Monthly Plot



Time Series Dashboard

Setter: Version 1 via rredis

```r
insertXtsR <- function(x, key) {
    xm <- coredata(x)
    xi <- as.integer(index(x))
    for (i in seq_len(nrow(xm))) {
        dat <- unname(c(xi[i], xm[i, , drop=TRUE]))
        redisRPush(key, dat)
    }
    invisible(NULL)
}
```

Time Series Dashboard

Getter: Base Version via rredis

```r
getXtsR <- function(key) {
    n <- as.integer(redisLLen(key))
    vals <- redisLRange(key, 0, n)
    m <- length(vals)
    mat <- matrix(NA, n, 8)
    dat <- rep(NA, n)
    for (i in 1:n) {
        z <- vals[[i]]
        dat[i] <- z[1]
        mat[i, ] <- z[-1]
    }
    x <- xts(mat, order.by=as.Date(dat, origin="1970-01-01"))
    colnames(x) <- colnams
    x
}
```

Time Series Dashboard

Getter: Rcpp Version 1

```r
getXtsRcpp1 <- function(key) {
    n <- as.integer(redis$llen(key))
    vals <- redis$lrange(key, 0, n)
    m <- length(vals)
    mat <- matrix(NA, n, 8)
    dat <- rep(NA, n)
    for (i in 1:n) {
        z <- vals[[i]]
        dat[i] <- z[1]
        mat[i, ] <- z[-1]
    }
    x <- xts(mat, order.by=as.Date(dat, origin="1970-01-01"))
    colnames(x) <- colnams
    x
}
```

Time Series Dashboard

Getter: Rcpp Version 2

```r
getXtsRcpp2 <- function(key) {
    mat <- redis$listToMatrix(redis$lrange(key, 0, -1))
    x <- xts(mat[,-1], order.by=as.Date(mat[,1], origin="1970-01-01"))
    colnames(x) <- colnams
    x
}
```

Time Series Dashboard

Timings

```
key <- "quandl:cme:sp1"
res <- benchmark(getXtsR(key),
                 getXtsRcpp1(key),
                 getXtsRcpp2(key),
                 order="relative", replications=25)[,1:4]
print(res)
```

```
##                test replications elapsed relative
## 3 getXtsRcpp2(key)           25   0.608    1.000
## 2 getXtsRcpp1(key)           25   1.768    2.908
## 1     getXtsR(key)           25  29.063   47.801
```

Time Series Dashboard

Can we do better?

- Yes: Redis also offers a binary type
- We grab each data row as a vector
- Pointer plus length a common form of expression

Time Series Dashboard

New Rcpp Function: R Side

```r
insertXtsRcpp <- function(x, key) {
    xm <- coredata(x)
    xi <- as.numeric(index(x))
    dat <- unname(cbind(xi, xm))
    for (i in seq_len(nrow(xm))) {
        redis$listRPush(key, dat[i,])
    }
    invisible(NULL)
}
```

Time Series Dashboard

New Rcpp Function: Setter

```
// redis "append to list" -- without R serialization
std::string listRPush(std::string key, Rcpp::NumericVector x) {

    // uses binary protocol, see hiredis docs
    redisReply *reply =
        static_cast<redisReply*>(redisCommand(prc_, "RPUSH %s %b",
                                              key.c_str(),
                                              x.begin(), x.size()*szdb));

    std::string res = "";
    freeReplyObject(reply);
    return(res);
}
```

Time Series Dashboard

New Rcpp Function: Getter

```
// redis "get from list from start to end" -- without R serialization
Rcpp::List listRange(std::string key, int start, int end) {
    redisReply *reply =
        static_cast<redisReply*>(redisCommand(prc_, "LRANGE %s %d %d",
                                              key.c_str(), start, end));
    checkReplyType(reply, replyArray_t); // ensure we got array
    unsigned int len = reply->elements;
    Rcpp::List x(len);
    for (unsigned int i = 0; i < len; i++) {
        checkReplyType(reply->element[i], replyString_t); // ensure binary
        int nc = reply->element[i]->len;
        Rcpp::NumericVector v(nc/szdb);
        memcpy(v.begin(), reply->element[i]->str, nc);
        x[i] = v;
    }
    freeReplyObject(reply);
    return(x);
}
```

Time Series Dashboard

Use This Way

```r
getXtsRcpp3 <- function(key) {
    mat <- redis$listToMatrix(redis$listRange(key, 0, -1))
    x <- xts(mat[,-1], order.by=as.Date(mat[,1], origin="1970-01-01"))
    colnames(x) <- colnams
    x
}
```

Time Series Dashboard

Timings

```
key2 <- "quandl:cme:sp1:rcpp"
res2 <- benchmark(getXtsR(key),
                  getXtsRcpp1(key),
                  getXtsRcpp2(key),
                  getXtsRcpp3(key2),
                  order="relative", replications=25)[,1:4]
print(res2)
```

```
##                test replications elapsed relative
## 4 getXtsRcpp3(key2)           25   0.364    1.000
## 3  getXtsRcpp2(key)           25   0.582    1.599
## 2  getXtsRcpp1(key)           25   1.747    4.799
## 1      getXtsR(key)           25  29.481   80.992
```

Time Series Dashboard

Status

- Not so bad: $\sim$ 80-fold increase for RcppRedis over rredis
- Inner retrieval (outside of `xts` creation) about 100 times faster
- 25 retrieval in 364 msec is clearly 'good enough'
- Limitation: Storing small binary vectors not elegant
- Possible fix: MessagePack
- Alternative to 'binary JSON' and alternative
- Easy to use API

Time Series Dashboard

Simple MessagePack buffer creation, then sending MessagePack buffer as binary load.

```
typedef msgpack::type::tuple<double, int, int, int> msg_t;

msgpack::sbuffer buffer;
msg_t m(v[0], (int)v[1], (int)v[2], (int)v[3]);     // fill the message type
msgpack::pack(buffer, m);                           // and pack it

replynew =
   static_cast<redisReply*>(redisCommand(d, "RPUSH %s %b",
                                         key.c_str(),
                                         buffer.data(), buffer.size()));
freeReplyObject(replynew);
```

Time Series Dashboard

Conclusion

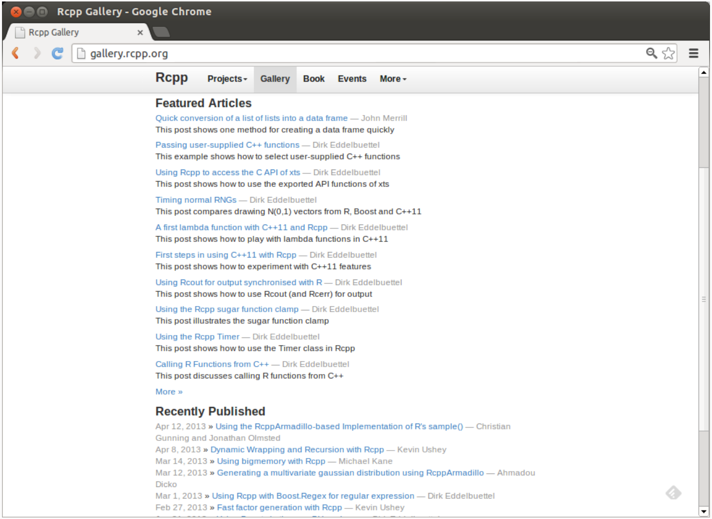
- Simple things remain simple
- Memory allocation, loops, conversions, … faster in C++
- Yet easily accessible from R
- Leverage R strength (eg shiny) by overcoming bottlenecks
- Leads to *Seamless Integration of R and C++* for accelerated modeling

The End

Documentation

- The Rcpp package comes with nine pdf vignettes, and numerous help pages.
- The introductory vignettes are now published (for Rcpp and RcppEigen in *J Stat Software*, for RcppArmadillo in *Comp Stat & Data Anlys*)
- The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has over 900 posts too, and And
- A number of blog posts introduce/discuss features.

Rcpp Gallery



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

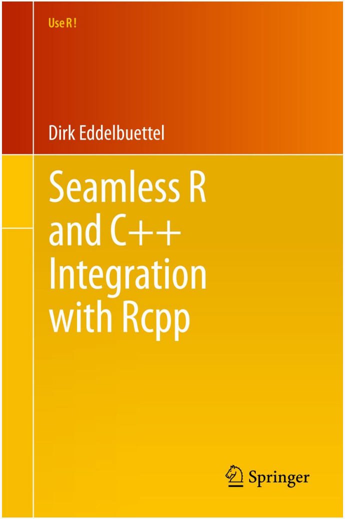
Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

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

The Rcpp book



Very Last Words

Thank You!

dirk@eddelbuettel.com