Rcpp

An Example-Driven Hands-on Introduction

Dirk Eddelbuettel

EARL 2015 Pre-Conference Workshop

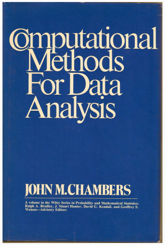
September 14, 2015

Introduction

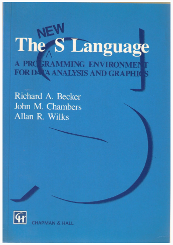
Brief Overview

- Preliminary: Motivation
- Part I: How To Get Started, Some Background
- Part II: Using Rcpp with Packages
- Part III: A Worked Rcpp Package Example
- Part IV: Some More Examples
- Brief Conclusion
- Appendix: C++ Refresher

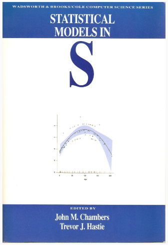
Motivation: R for *Programming with Data*



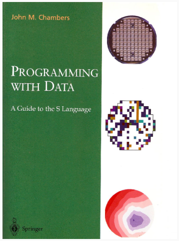
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.



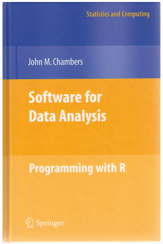
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



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



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



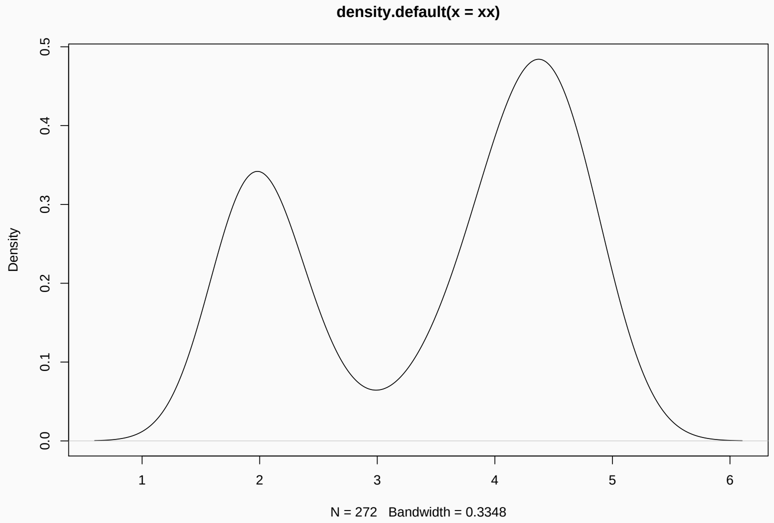
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

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

A Simple Example

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

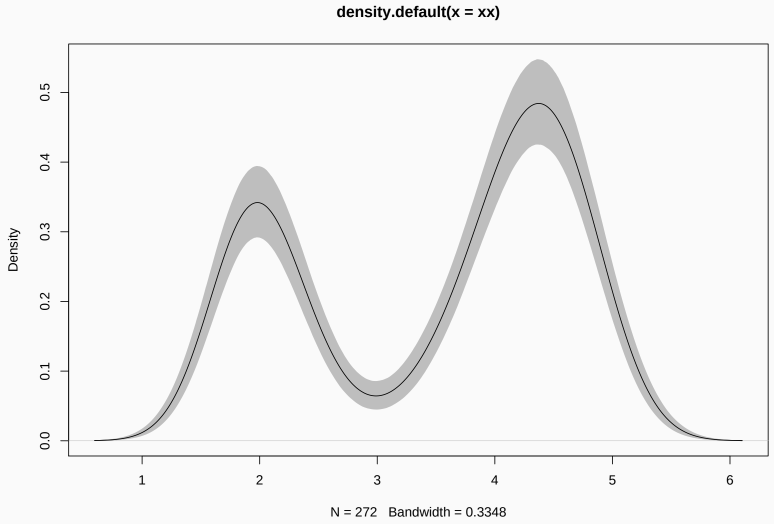
A Simple Example



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 the new *Reference Classes* now in R and built upon via Rcpp Modules, Rcpp Classes (and also RcppR6)

Why C++?

- Asking Google leads to about 52 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*

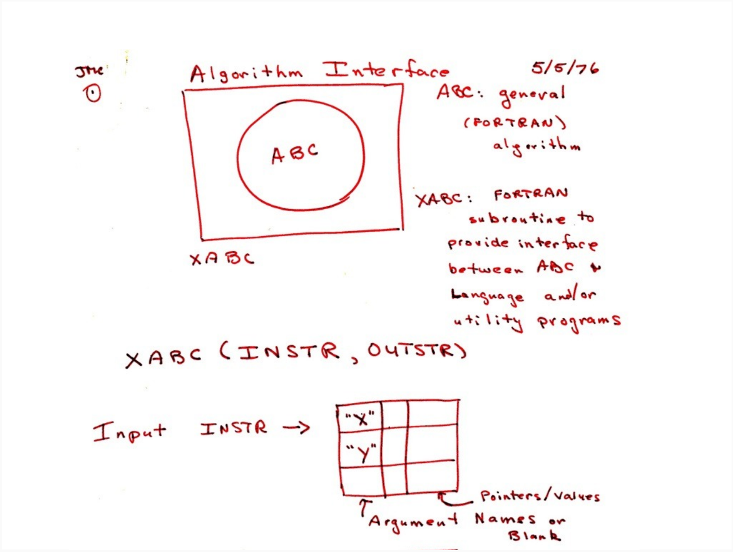
- *C* provides a rich inheritance and interoperability as Unix, Windows, … are all build on C.
- *Object-Oriented C++* (maybe just to provide endless discussions about exactly what OO is or should be)
- *Templated C++* which is mighty powerful; template meta programming unequalled in other languages.
- *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

Vision at Bell Labs, May 1976



JMC ①

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 →

ABC
XABC
"X"
"Y"
Pointers/Values
Argument Names or Blank

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

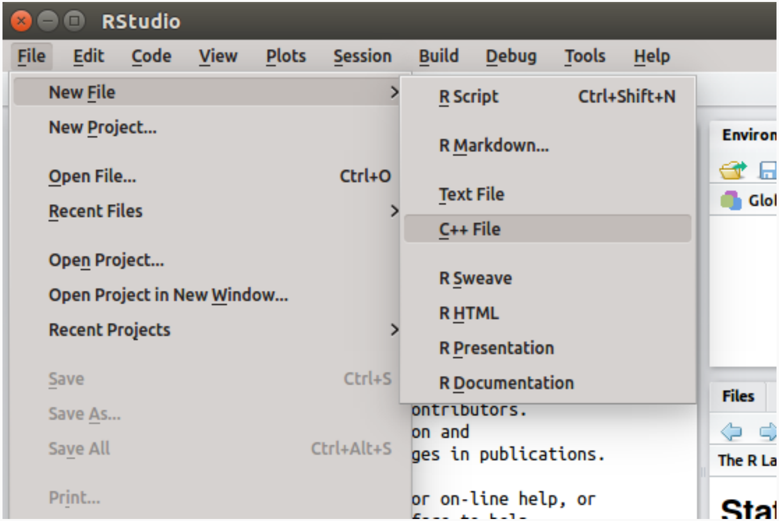
Why Rcpp?

- *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*

Part I: 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)
- Then at the R prompt:

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

Excercise

Building on the previous example, build a function `adder`

- that adds two vectors
- hint: as we will see see later, + is vectorised

What other features might such a function need?

One solution:

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

// [[Rcpp::export]]
NumericVector adder(NumericVector x,
                    NumericVector y) {
  NumericVector z = x + y;
  return z;
}
```

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
```

cppFunction Exercise

Create a function or two using `cppFunction()` supplying an argument string with the code.

Experiment.

Use *e.g.* `IntVector` and/or `NumericVector`.

One solution:

```
cppFunction('double myrange(NumericVector(x)) {
                return max(x) - min(x);
              }')
```

Linebreaks above purely stylistic. String can be multiline too, use ‘paste()‘ etc pp.

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: A Motivating Example

Consider a function defined as

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

Simple R Implementation

R implementation and use:

```r
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
```

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   0.018    1.000
## 2 F(15)          100   0.200   11.111
## 3 F(20)          100   2.167  120.389
```

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)); }')
## Using it on first 11 arguments
sapply(0:10, G)
```

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

An Introductory Example: Comparing timing

Timing:

```
library(rbenchmark)
benchmark(F(25), G(25))[,1:4]
```

```
##     test replications elapsed relative
## 1 F(25)          100  24.373  518.574
## 2 G(25)          100   0.047    1.000
```

A nice gain of a few orders of magnitude.

Background: R Framework

Everything uses this interface

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

which is explained in *Writing R Extensions*.

But we **never** have to deal with `.Call()` function, or the `SEXP` types.

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:

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

As we saw before: vectorized C++!

STL Type mapping

Use of `std::vector<double>` and STL algorithms:

```cpp
#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;
}
```

STL Type mapping

Used via

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

Type mapping is seamless

Outer product of a column vector (via RcppArmadillo):

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

Uses implicit conversion – see vignette Rcpp-extending.

In C++ source files use the attribute:

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

C++11: LAMBDAS, AUTO, AND MUCH MORE

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;
}
```

Part II: Rcpp and Packages

Basic Usage: Packages

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

They allow us to bundle

- code
- (optional) documentation
- (optional) data
- (optional) tests

and made development, versioning, sharing, … easy thanks to the R infrastructure.

Basic Usage: Packages

As of mid September 2015, there are

- 461 packages on CRAN which use Rcpp,
- a further 59 on BioConductor,
- an unknown number on GitHub

So well over 500 packages on the official, tested repositories – all with working, tested, and reviewed examples.

Basic Usage: Packages

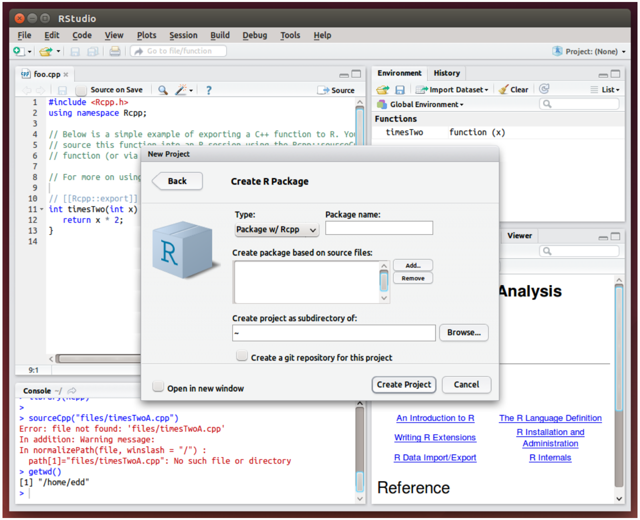
Creating packages with Rcpp is easy!

`Rcpp.package.skeleton()` creates an empty package.

The vignette Rcpp-packages has fuller details.

Packages and Rcpp

Or just use RStudio (similar to `Rcpp.package.skeleton()`)



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

// Below is a simple example of exporting a C++ function to R. Yo
// source this function into an R session using the Rcpp::sourceC
// function (or via

// For more on using

// [[Rcpp::export]]
int timesTwo(int x)
    return x * 2;
}
```

Functions

timesTwo	function (x)

New Project

Back

Create R Package

Type: Package w/ Rcpp

Package name:

Create package based on source files:

Add...

Remove

Create project as subdirectory of: ~

Browse...

Create a git repository for this project

Open in new window

Create Project

Cancel

```
> library(Rcpp)
>
> 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"
>
```

Analysis

An Introduction to R

The R Language Definition

Writing R Extensions

R Installation and Administration

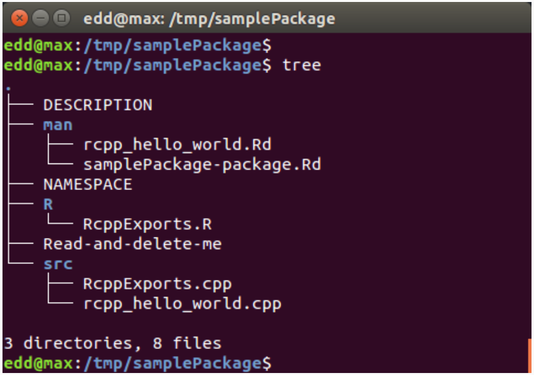
R Data Import/Export

R Internals

Reference

Packages and Rcpp: Files Created

These files are created by `Rcpp.package.skeleton()`



```
edd@max:/tmp/samplePackage$
edd@max:/tmp/samplePackage$ tree
.
├── DESCRIPTION
├── man
│   ├── rcpp_hello_world.Rd
│   └── samplePackage-package.Rd
├── NAMESPACE
├── R
│   └── RcppExports.R
├── Read-and-delete-me
└── src
    ├── RcppExports.cpp
    └── rcpp_hello_world.cpp

3 directories, 8 files
edd@max:/tmp/samplePackage$
```

We will examine some of the files. Using the RStudio feature is very similar. You can use either.

Packages and Rcpp Files: DESCRIPTION

```
Package: samplePackage
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2015-09-06
Author: Your Name
Maintainer: Your Name <your@email.com>
Description: More about what it does (maybe more than one
License: GPL (>= 2)
Imports: Rcpp (>= 0.12.0.5)
LinkingTo: Rcpp
```

Here the last two lines are key: we **import** Rcpp and **link to** it.

Packages and Rcpp Files: NAMESPACE

```
useDynLib(samplePackage)
exportPattern("^[[:alpha:]]+")
importFrom(Rcpp, evalCpp)
```

This ensures that

- we load the code in this package
- export all (public) identifiers
- load a function from Rcpp (to ensure Rcpp gets imported)

Packages and Rcpp Files: src/rcpp_hello_world.cpp

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

// [[Rcpp::export]]
List rcpp_hello_world() {
  CharacterVector x =
       CharacterVector::create("foo", "bar");
  NumericVector y   =
       NumericVector::create(0.0, 1.0);
  List z            = List::create(x, y);
  return z;
}
```

Files R/RcppExports.R, src/RcppExports.cpp

These files are *auto-generated* by the function `compileAttributes()`.

Whenever we change the "interface" from our C++ function (that is exported to R) we need to re-run `compileAttributes()` – which RStudio automates.

Exercise: Package

Build and load the package.

Check the package.

Add a (C++) function to the package.

Packages: Rcpp and RcppArmadillo

Create packages via `Rcpp.package.skeleton()` and `RcppArmadillo.package.skeleton()`:

```
// [[Rcpp::depends(RcppArmadillo)]]
//
// another simple example: outer product of a vector, returns 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;
}
```

RcppArmadillo is fabulous

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

// [[Rcpp::export]]
List fastLm(const arma::mat& X, const arma::colvec& y) {
    int n = X.n_rows, k = X.n_cols;

    arma::colvec coef = arma::solve(X, y);     // fit model y ~ X
    arma::colvec res  = y - X*coef;            // residuals

    // std.errors of coefficients
    double s2 =
      std::inner_product(res.begin(), res.end(), res.begin(), 0.0)/(n - k);

    arma::colvec std_err =
      arma::sqrt(s2 * arma::diagvec(arma::pinv(arma::trans(X)*X)));

    return List::create(_["coefficients"] = coef,
                        _["stderr"]       = std_err,
                        _["df.residual"]  = n - k    );
}
```

Other Rcpp Integrations

RcppEigen has its fans, notably lme4.

RcppGSL was just updated, allows easy access to the GSL.

And *your favourite package here* as it is easy to write plugins, and even easier to just build against a library.

Part III: RcppZiggurat

Motivation

Random numbers play an ever increasing role as simulations are so widely used.

R contains several high-quality generators.

Other languages, however, also have *fast* ones.

Past

In 2000, Marsaglia and Tsang introduced Ziggurat: a new and *fast* generator for $N(0,1)$ draws.

In 2005, Leong, Zhang et al improved one aspect. This is the version we use.

These were however done only for 32 bit machines.

Our RcppZiggurat package improves on that aspect, compares multiple implementations and provides a new easy-to-use version.

Basic Code

From the Marsaglia and Tsang (2000) version:

```
#include <math.h>
static unsigned long jz,jsr=123456789;

#define SHR3 (jz=jsr, jsr^=(jsr<<13), jsr^=(jsr>>17), jsr^=(jsr<<5),jz+jsr)
#define UNI (.5 + (signed) SHR3*.2328306e-9)
#define IUNI SHR3

static long hz;
static unsigned long iz, kn[128], ke[256];
static float wn[128],fn[128], we[256],fe[256];

#define RNOR (hz=SHR3, iz=hz&127, (fabs(hz)<kn[iz])? hz*wn[iz] : nfix())
```

A handful of (global) variables and a handful of macros.

Plus a setup function and the `nfix()` tail fix.

Basic Code

From the Leong et al (2005) version:

```
#define MWC  ((znew<<16)+wnew )
#define SHR3 (jz=jsr, jsr^=(jsr<<13), jsr^=(jsr>>17), jsr^=(jsr<<5),jz+jsr)
#define CONG (jcong=69069*jcong+1234567)
#define KISS ((MWC^CONG)+SHR3)

#define RNOR (hz=KISS, iz=hz&127, (fabs(hz)<kn[iz]) ? hz*wn[iz] : nfix())
```

Five macros, plus the same two functions.

Basic Issue

Macros.

Not exactly *de rigeur* any more.

Maybe once defensible for performance but notorious for possible side-effects.

Alternative? A C++ Class!

```
class ZigguratLZLLV : public Zigg {
public:
    ZigguratLZLLV(uint32_t seed=123456789) :
        jsr(123456789), z(362436069), w(521288629), jcong(380116160)  {
        setup();
        setSeed(seed);
    }
    inline double norm(void) { return RNOR; }
    void setSeed(const uint32_t jsrseed) { /* ... */ }
    uint32_t getSeed() { /* ... */ }

private:
    uint32_t jz, jsr, z, w, jcong;
    int32_t hz;
    uint32_t iz, kn[128];
    double wn[128],fn[128];

    void setup(void) { /* ... */ }
    inline double nfix(void) { /* ... */ }
};
```

Better Design!

This allows us

- keep all state variables private
- import and export only the seed
- but still have access to the trusted code
- and use only headers not requiring linking.

See the RcppZiggurat package, and its vignette, for much, much more detail.

SAMPLECODE AND ITS EARL-2015-09 DIRECTORY

We maintain a catch-all repository samplecode on GitHub.

It contains a subdirectory `earl-2015-09` with a complete package UseZiggurat.

SAMPLECODE AND ITS EARL-2015-09 DIRECTORY

The package contains (essentially) two source files, and an entry in `DESCRIPTION` about where to find the required resources.

Via `LinkingTo: RcppZiggurat`, we tell R to let us access the *header files* of the named package.

This is sufficient as the design from the previous section can be implements in headers only.

callZiggurat

This function calls the ZigguratLZLLV generator – our implementation of the 2005 JSS paper “gold standard”.

```
#include <Rcpp.h>
#include <ZigguratLZLLV.h>

static Ziggurat::LZLLV::ZigguratLZLLV z;

// [[Rcpp::export]]
bool setZigguaratSeed(const int seed) {
    z.setSeed(seed);
    return true;
}

// [[Rcpp::export]]
Rcpp::NumericVector callZiggurat(int n) {
    Rcpp::NumericVector X(n);
    for (int i=0; i<n; i++) X(i) = z.norm();   // Ziggurat draw
    return X;
}
```

CALLRNORM

For comparison, we call `rnorm` from R (via its Rcpp variant)

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector callrnorm(int n) {
    // have to call set.seed at the R level ...
    return Rcpp::rnorm(n);
}
```

Here the seed setting is done from R rather than via an argument...

Comparison

Code from `demo/zigguratVsR.R`

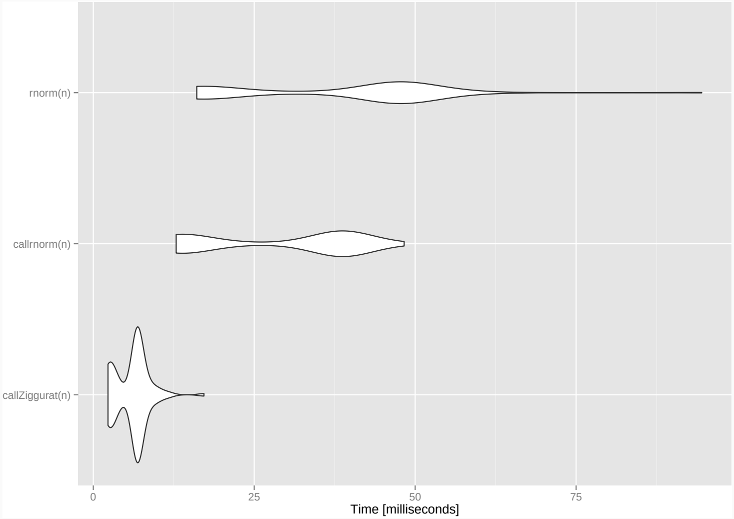
```r
library(UseZiggurat)
library(microbenchmark)

seed <- 12345
n <- 1e5
N <- 100

set.seed(seed)
setZigguratSeed(seed)

res <- microbenchmark(callZiggurat(n), callrnorm(n), rnorm(n), times = N)
```

Comparison

Microbenchmark result:

```
## Unit: milliseconds
##             expr   min     lq   mean median     uq   max neval cld
##  callZiggurat(n)  2.25  3.474  6.014  6.781  6.908 17.23   100 a
##     callrnorm(n) 12.66 13.529 28.484 37.879 38.340 48.51   100  b
##         rnorm(n) 15.51 17.725 37.502 46.680 47.753 95.08   100   c
```

Comparison



rnorm(n)
callrnorm(n)
callZiggurat(n)
0
25
50
75
Time [milliseconds]

Part IV: Further 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.

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

```cpp
#include <Rcpp.h>

using namespace Rcpp;

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

/*** R
callFunction(x, fivenum)
callFunction(x, summary)
*/
```

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

```cpp
#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

```r
Rcpp::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
# R gets the same results (in reverse)
# and also returns the eigenvectors.
```

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

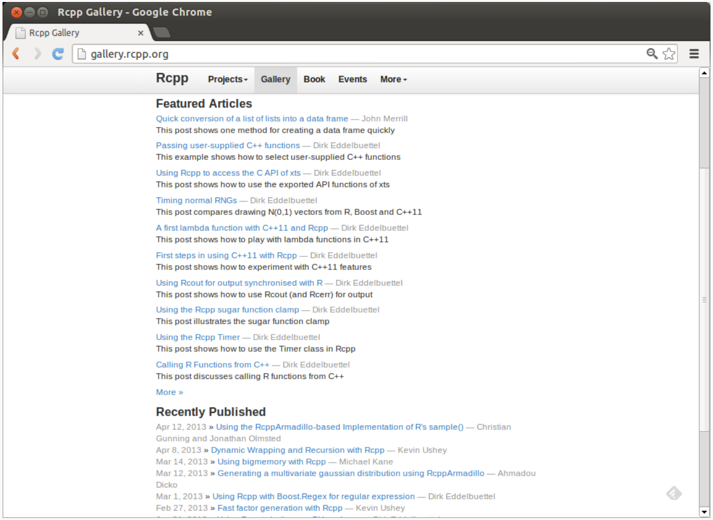
```cpp
#include <Rcpp.h>
using namespace Rcpp;

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;
}
```

The End

Documentation

- The 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 a fair number of posts too.
- 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

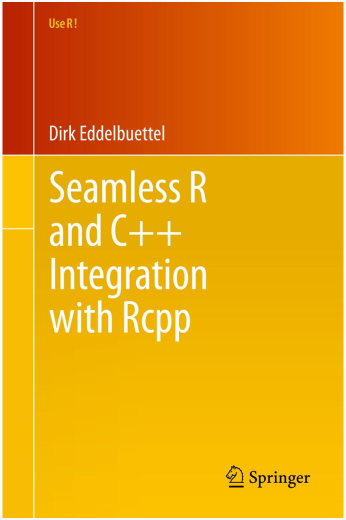
Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

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


Use R!

Dirk Eddelbuettel

Seamless R and C++ Integration with Rcpp

Springer

Very Last Words

Thank You!

`dirk@eddelbuettel.com`

Appendix: C++ Primer

Programming with C++

- C++ Basics
- Debugging
- Best Practices

and then on to Rcpp itself

Compiled not Interpreted

Need to compile and link

```
#include <cstdio>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

Compiled not Interpreted

Or streams output rather than `printf`

```
#include <iostream>

int main(void) {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Compiled not Interpreted

`g++ -o` will compile and link

We will now look at an examples with explicit linking.

Compiled not Interpreted

```
#include <cstdio>

#define MATHLIB_STANDALONE
#include <Rmath.h>

int main(void) {
    printf("N(0,1) 95th percentile %9.8f\n",
           qnorm(0.95, 0.0, 1.0, 1, 0));
}
```

Compiled not Interpreted

We may need to supply:

- *header location* via `-I`,
- *library location* via `-L`,
- *library* via `-llibraryname`

```
g++ -I/usr/include -c qnorm_rmath.cpp
g++ -o qnorm_rmath qnorm_rmath.o -L/usr/lib -lRmath
```

Statically Typed

- R is dynamically typed: `x <- 3.14; x <- "foo"` is valid.
- In C++, each variable must be declared before first use.
- Common types are `int` and `long` (possibly with `unsigned`), `float` and `double`, `bool`, as well as `char`.
- No standard string type, though `std::string` is close.
- All these variables types are scalars which is fundamentally different from R where everything is a vector.
- `class` (and `struct`) allow creation of composite types; classes add behaviour to data to form `objects`.
- Variables need to be declared, cannot change

C++ is a Better C

- control structures similar to what R offers: `for`, `while`, `if`, `switch`
- functions are similar too but note the difference in positional-only matching, also same function name but different arguments allowed in C++
- pointers and memory management: very different, but lots of issues people had with C can be avoided via STL (which is something Rcpp promotes too)
- sometimes still useful to know what a pointer is …

Object-Oriented

This is a second key feature of C++, and it does it differently from S3 and S4.

```
struct Date {
    unsigned int year;
    unsigned int month;
    unsigned int day
};

struct Person {
    char firstname[20];
    char lastname[20];
    struct Date birthday;
    unsigned long id;
};
```

Object-Oriented

Object-orientation in the C++ sense matches data with code operating on it:

```
class Date {
private:
    unsigned int year
    unsigned int month;
    unsigned int date;
public:
    void setDate(int y, int m, int d);
    int getDay();
    int getMonth();
    int getYear();
}
```

Generic Programming and the STL

The STL promotes *generic* programming.

For example, the sequence container types `vector`, `deque`, and `list` all support

- `push_back()` to insert at the end;
- `pop_back()` to remove from the front;
- `begin()` returning an iterator to the first element;
- `end()` returning an iterator to just after the last element;
- `size()` for the number of elements;

but only `list` has `push_front()` and `pop_front()`.

Other useful containers: `set`, `multiset`, `map` and `multimap`.

Generic Programming and the STL

Traversal of containers can be achieved via *iterators* which require suitable member functions `begin()` and `end()`:

```
std::vector<double>::const_iterator si;
for (si=s.begin(); si != s.end(); si++)
    std::cout << *si << std::endl;
```

Generic Programming and the STL

Another key STL part are *algorithms*:

```
double sum = accumulate(s.begin(), s.end(), 0);
```

Some other STL algorithms are

- `find` finds the first element equal to the supplied value
- `count` counts the number of matching elements
- `transform` applies a supplied function to each element
- `for_each` sweeps over all elements, does not alter
- `inner_product` inner product of two vectors

Template Programming

Template programming provides a ‘language within C++’: code gets evaluated during compilation.

One of the simplest template examples is

```
template <typename T>
const T& min(const T& x, const T& y) {
    return y < x ? y : x;
}
```

This can now be used to compute the minimum between two `int` variables, or `double`, or in fact any *admissible type* providing an `operator<()` for less-than comparison.

Template Programming

Another template example is a class squaring its argument:

```
template <typename T>
class square : public std::unary_function<T,T> {
public:
    T operator()(T t) const {
        return t*t;
    }
};
```

which can be used along with STL algorithms:

```
transform(x.begin(), x.end(), square);
```

Further Reading

Books by Meyers are excellent

I also like the (free) C++ Annotations

C++ FAQ

Resources on StackOverflow such as

- general info and its links, eg
- booklist

Collophon

Made using

- TeXlive 20141024
- Beamer with mtheme
- Pandoc 1.12.4.2
- R 3.2.2
- rmarkdown 0.7
- Emacs 24.4
- Ubuntu 15.04