Rcpp

An Example-Driven Hands-on Introduction

Dirk Eddelbuettel

EARL 2015 Pre-Conference Workshop

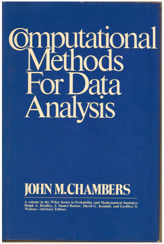
November 2, 2015

Introduction

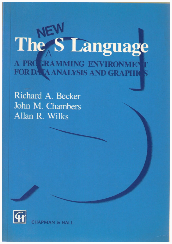
Brief Overview

- 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: A Worked ‘Faster Application’ Example
- Part V: Lots More (Small) Examples
- Brief Conclusion
- Appendix: C++ Refresher

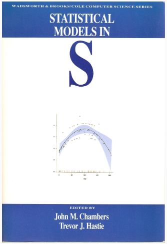
Motivation: R for *Programming with Data*



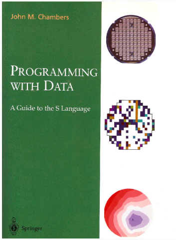
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.



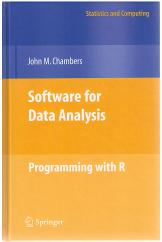
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



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



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



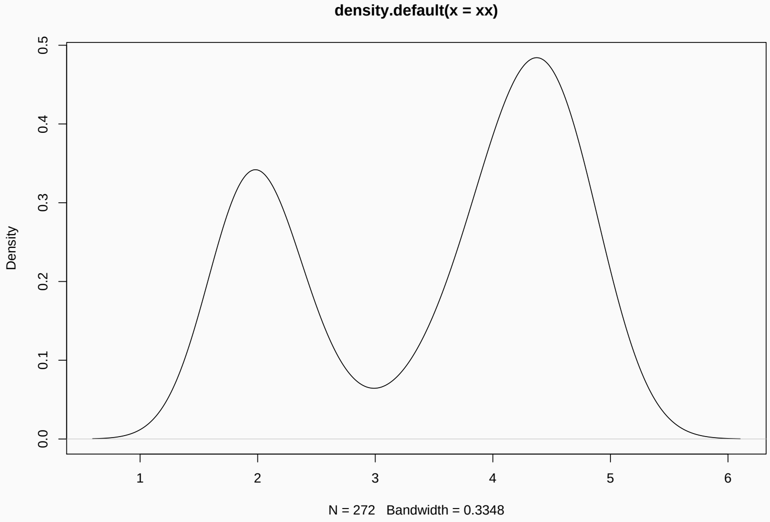
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

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)
```

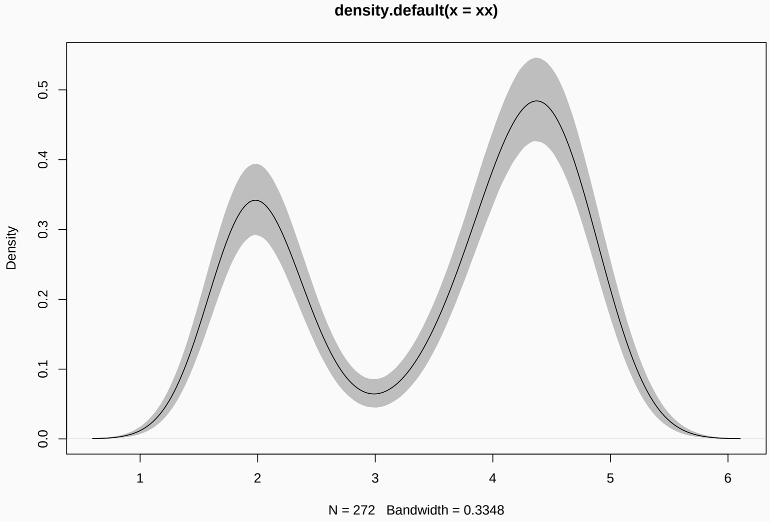
A Simple Example



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*

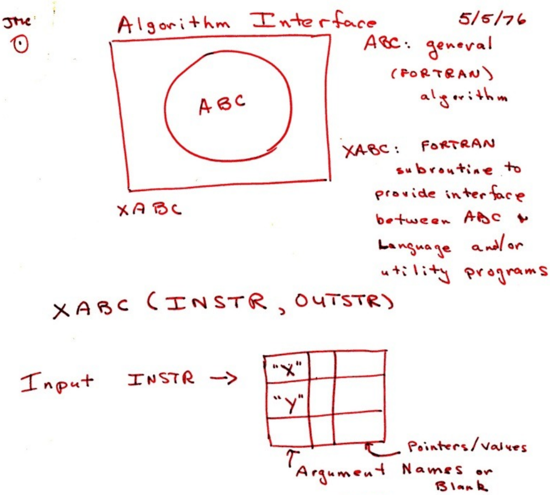
- *C* provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.
- *Object-Oriented C++* (maybe just to provide endless discussions about exactly what OO is or should be)
- *Templated C++* which is mighty powerful; template meta programming unequalled in other languages.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



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

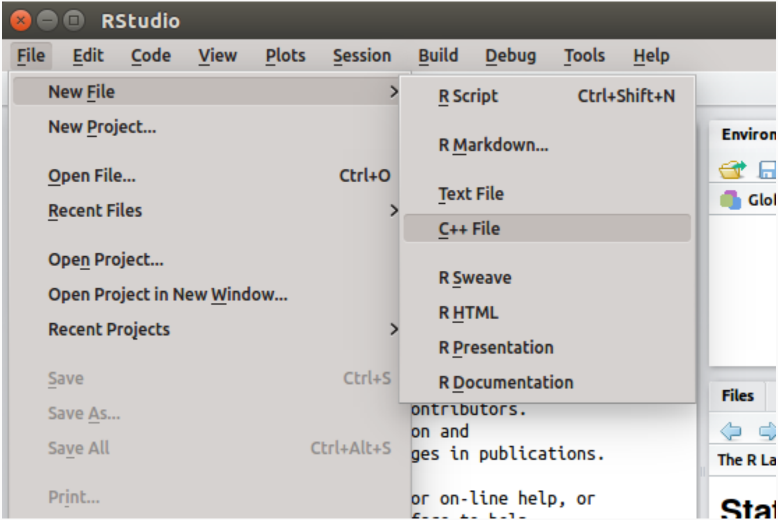
Why Rcpp?

- *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:



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:

```
## 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:

```
#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.

```
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.* `IntegerVector` 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()` 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
```

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:

```
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   0.019    1.000
## 2 F(15)          100   0.198   10.421
## 3 F(20)          100   2.208  116.211
```

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  24.496  510.333
## 2 G(25)          100   0.048    1.000
```

A nice gain of a few orders of magnitude.

Fibonacci Exercise

Copy one or two of the Fibonacci solutions and excercise.

Install the rbenchmark or microbenchmark package and play with timings.

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:

```
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:

```
#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

```
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 early November 2015, there are

- 503 packages on CRAN which use Rcpp,
- a further 70 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

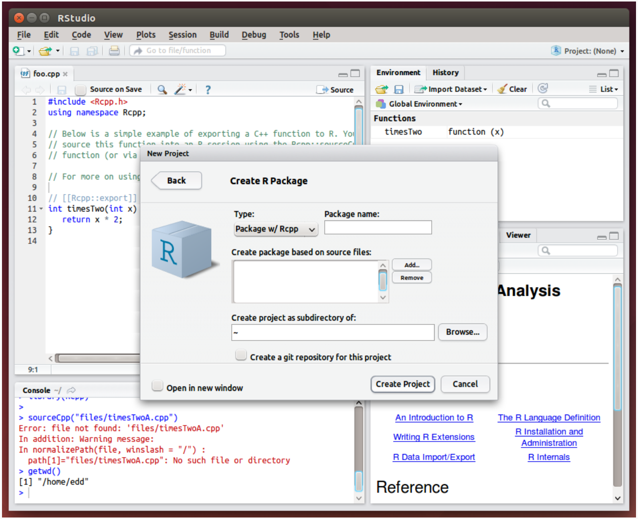
Creating packages with Rcpp is easy!

`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()`)



Packages and Rcpp: Files Created

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

```
edd@max: /tmp/samplePackage
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

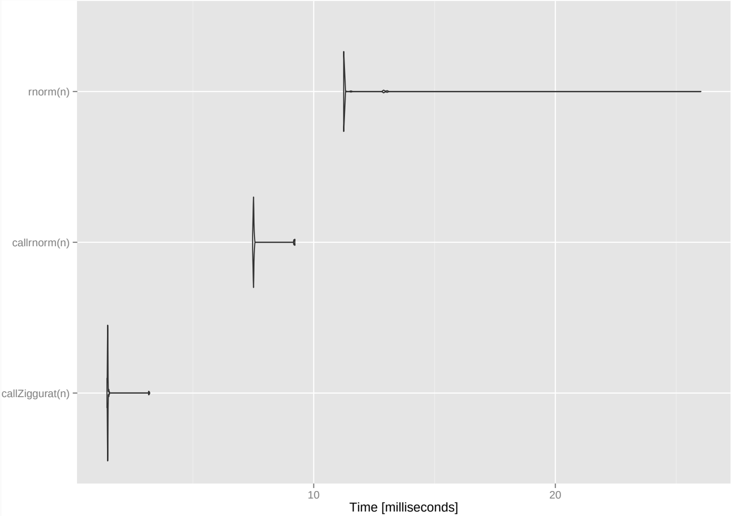
Code from demo/zigguratVsR.R

```
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)
```

Microbenchmark result:

```
## Unit: milliseconds
##              expr    min     lq   mean median     uq    max neval cld
##  callZiggurat(n)  1.439  1.466  1.524  1.474  1.480  3.216   100 a
##     callrnorm(n)  7.465  7.497  7.626  7.509  7.522  9.225   100  b
##         rnorm(n) 11.187 11.213 11.464 11.227 11.241 26.070   100   c
```

Comparison



Part IV: RcppRedis

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

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)

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

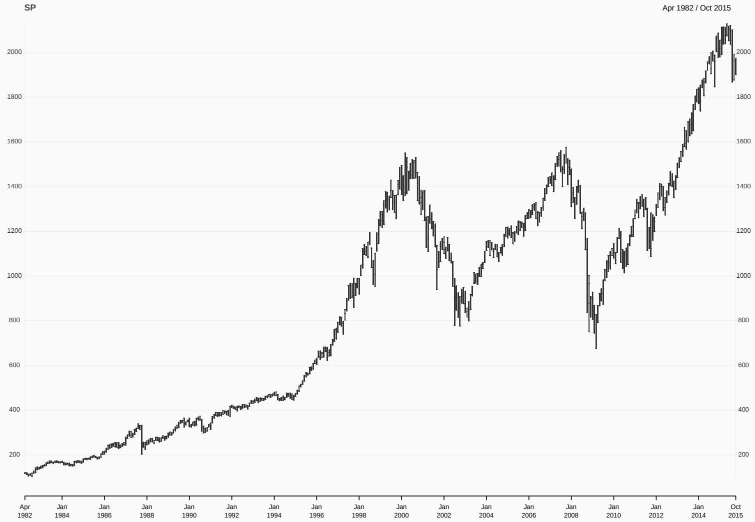
What is wrong with that?

- 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

Getting Data

```
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



Setter: Version 1 via rredis

```
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)
}
```

Getter: Base Version via rredis

```
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
}
```

Getter: Rcpp Version 1

```
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
}
```

Getter: Rcpp Version 2

```
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
}
```

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

New Rcpp Function: R Side

```
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)
}
```

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);
}
```

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);
}
```

Use This Way

```
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
}
```

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
```

Status

- Not so bad: $\sim$ 80-fold increase for RcppRedis over rredis
- Inner retrievals (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

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);
```

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

Part V: Further Examples

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;
}
```

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++

```
#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

```
#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

```
#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++

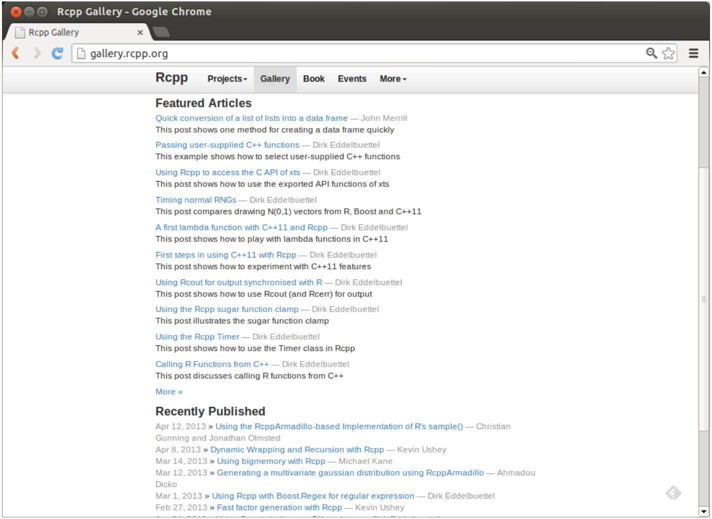
```
#include <Rcpp.h>
using namespace Rcpp;

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

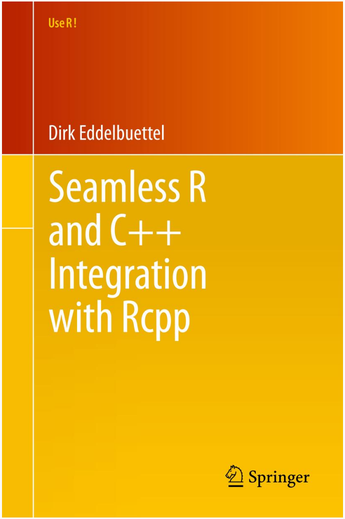
Documentation

- 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



The Rcpp book



Thank You!

dirk@eddelbuettel.com

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

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-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();
}
```

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`.

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;
```

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.

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 0.8.1
- Emacs 24.4
- Ubuntu 15.04