A Gentle and Applied Introduction to Rcpp

Dirk Eddelbuettel

Workshops for Ukraine

9 Feb 2023

ILLINOIS

https://dirk.eddelbuettel.com/papers/ua_workshop_rcpp_feb2023.pdf

Overview

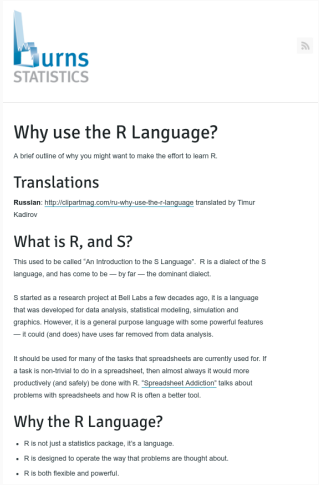
Very Broad Outline

Overview

- Motivation: Why R, Why Rcpp?
- Who Uses This?
- How Does One Use it?
- Usage Illustrations

Why R?

Why R? Pat Burn's View



Burns STATISTICS

Why use the R Language?

A brief outline of why you might want to make the effort to learn R.

Translations

Russian: http://clipartmag.com/ru-why-use-the-r-language translated by Timur Kadirov

What is R, and S?

This used to be called "An Introduction to the S Language". R is a dialect of the S language, and has come to be — by far — the dominant dialect.

S started as a research project at Bell Labs a few decades ago, it is a language that was developed for data analysis, statistical modelling, simulation and graphics. However, it is a general purpose language with some powerful features — it could (and does) have uses far removed from data analysis.

It should be used for many of the tasks that spreadsheets are currently used for. If a task is non-trivial to do in a spreadsheet, then almost always it would more productively (and safely) be done with R. "Spreadsheet Addiction" talks about problems with spreadsheets and how R is often a better tool.

Why the R Language?

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

Why the R Language?

Screen shot on the left part of short essay at Burns-Stat

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

The (much longer) R Inferno (free pdf, also paperback) is highly recommended.

Why R? Pat Burn's View

Why the R Language?

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

And more on why R as a language for data analysis.

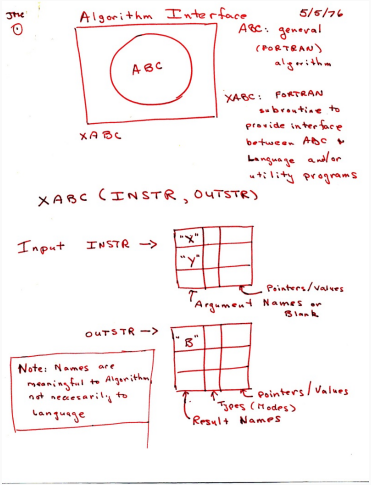
Source: https://www.burns-stat.com/documents/tutorials/why-use-the-r-language/

Why R?

R as a Powerful and Extensible Environment

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

Historical Perspective: R as 'The Interface'



A design sketch called 'The Interface'

AT&T Research lab meeting notes

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

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

Lead to "The Interface"

Which became S which lead to R

Source: John Chambers, personal communication; now also doi://10.1145/3386334

Historical Perspective: R as 'The Interface'

Proc ACM Progam Lang HOPL (History of Programming Languages) Paper

Chambers (2020) describes the fuller history of S and R, including the 'interface' sketch.

S, R, and Data Science

JOHN M. CHAMBERS, Stanford University, USA

Shepherd: Jean-Baptiste Tristan, Oracle Labs, USA

Data science is increasingly important and challenging. It requires computational tools and programming environments that handle big data and difficult computations, while supporting creative, high-quality analysis. The R language and related software play a major role in computing for data science. R is featured in most programs for training in the field. R packages provide tools for a wide range of purposes and users. The description of a new technique, particularly from research in statistics, is frequently accompanied by an R package, greatly increasing the usefulness of the description.

The history of R makes clear its connection to data science. R was consciously designed to replicate in open-source software the contents of the S software. S in turn was written by data analysis researchers at Bell Labs as part of the computing environment for research in data analysis and collaborations to apply that research, rather than as a separate project to create a programming language. The features of S and the design decisions made for it need to be understood in this broader context of supporting effective data analysis (which would now be called data science). These characteristics were all transferred to R and remain central to its effectiveness. Thus, R can be viewed as based historically on a domain-specific language for the domain of data science.

CCS Concepts: • **Software and its engineering → General programming languages**; • **Social and professional topics** → *History of programming languages*.

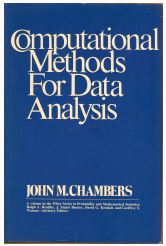
Additional Key Words and Phrases: data science, statistical computing, scientific computing

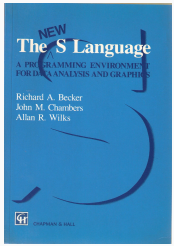
ACM Reference Format:
John M. Chambers. 2020. S, R, and Data Science. *Proc. ACM Program. Lang.* 4, HOPL, Article 84 (June 2020), 17 pages. https://doi.org/10.1145/3386334

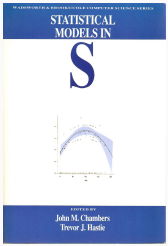
From Section 3.3:

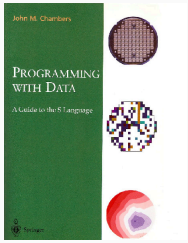
The Rcpp interface to C++ is used extensively in packages based on specialized C++ code. The original Rcpp is described in [Eddelbuettel and François 2011], but the interface has been much extended in the version now on CRAN. Approximately 10% of the packages on CRAN use Rcpp. Rcpp includes extensions to C++ to support a high-level programming style with R objects that in many ways resurrects the features of the original interface language of Section 2.3, but now for C++.

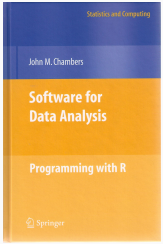
Why R? : *Programming with Data* from 1977 to 2016

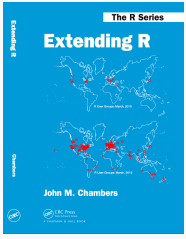






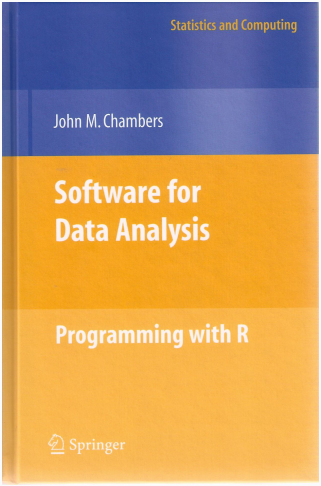






Thanks to John Chambers for high-resolution cover images. The publication years are, respectively, 1977, 1988, 1992, 1998, 2008 and 2016.

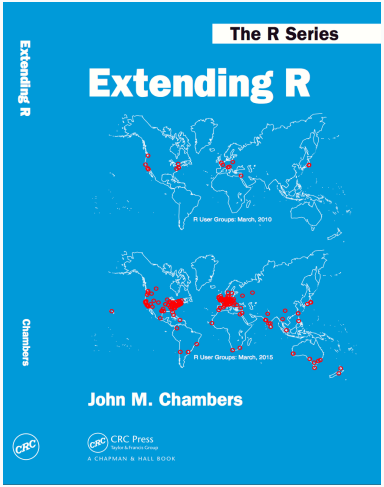
Chambers (2008)



Software For Data Analysis

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

Chambers (2016)



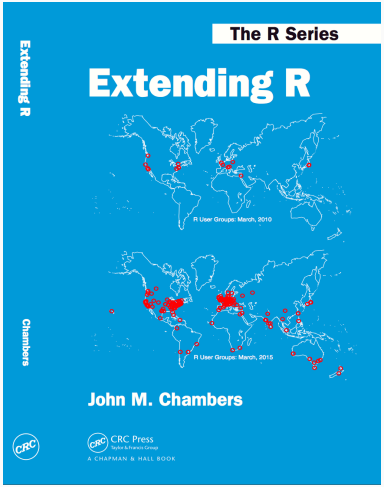
Extending R

Object: Everything that exists in R is an object

Function: Everything that happens in R is a function call

Interface: Interfaces to other software are part of R

Chambers (2016)



Extending R, Chapter 4

The fundamental lesson about programming in the large is that requires a correspondingly broad and flexible response. In particular, no single language or software system os likely to be ideal for all aspects. Interfacing multiple systems is the essence. Part IV explores the design of of interfaces from R.

C++ and Rcpp for *Extending R*

A good fit, it turns out

- A good part of R is written in C (besides R and Fortran code)
- The principle interface to external code is a function `.Call()`
- It can call functions we write which adhere to the interface
  - that takes one or more of the high-level `SEXP` data structures R uses
  - and returns one `SEXP`
- Formally a function named, say, `myfunc`, will use

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

A good fit, it turns out (cont.)

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

Other good reasons

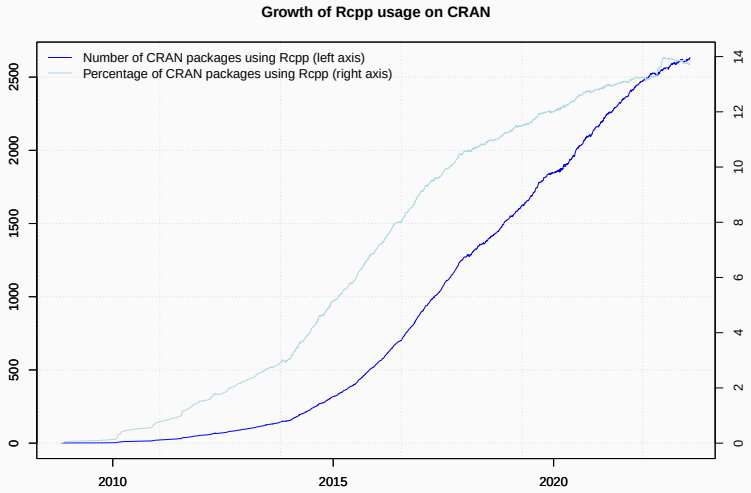
- It is *fast* – compiled C++ is hard to beat in other languages
  - (That said, you can *of course* write bad and slow code....)
- It is *very general* and widely used
  - *many libraries*
  - many tools
- It is fairly universal:
  - just about anything will have C interface so C++ can play
  - just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

Key Features

- (Fairly) Easy to learn as it really does not have to be complicated – many examples
- Easy to use as it avoids build / OS system complexities thanks to R infrastrucure
- Expressive as it allows for *vectorised* C++ using *Rcpp Sugar*
- Seamless R object access: vector, matrix, list, S3/S4, Environment, Function, ...
- Speed gains for ariety of tasks where R struggles: loops, function calls, ...
- Extensible facilitates access to external libraries directly or via eg *Rcpp modules*

Who Uses R?

Growth



Data current as of February 4, 2023.

Users on Core Repositories

Rcpp is currently used by

- 2631 CRAN packages
- 252 BioConductor packages
- an unknown (but “large”) number of GitHub projects

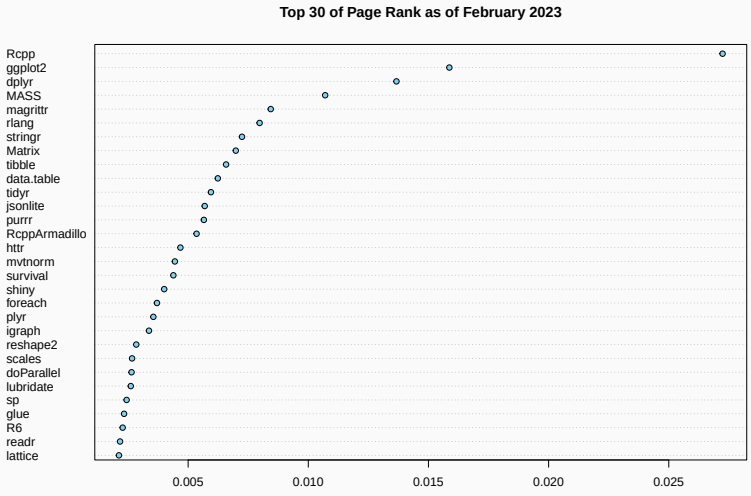
Pagerank

```
suppressMessages(library(utils))
library(pagerank)   # cf github.com/andrie/pagerank

cran <- "https://cran.r-project.org"
pr <- compute_pagerank(cran)
round(100*pr[1:5], 3)
```

```
##     Rcpp  ggplot2    dplyr     MASS magrittr
##    2.724    1.587    1.367    1.070    0.844
```

Pagerank


Top 30 of Page Rank as of February 2023
Rcpp
ggplot2
dplyr
MASS
magrittr
rlang
stringr
Matrix
tibble
data.table
tidyr
jsonlite
purrr
RcppArmadillo
httr
mvtnorm
survival
shiny
foreach
plyr
igraph
reshape2
scales
doParallel
lubridate
sp
glue
R6
readr
lattice
0.005
0.010
0.015
0.020
0.025

Percentage of Compiled Packages

```
db <- tools::CRAN_package_db()   # added in R 3.4.0
db <- db[!duplicated(db[,1]),]   # rows: nb of pkgs,
nTot <- nrow(db)                 # cols: different attributes
nRcpp <- length(tools::dependsOnPkgs("Rcpp",recursive=FALSE, installed=db))
nCompiled <- table(db[, "NeedsCompilation"])["yes"]
propRcpp <- nRcpp / nCompiled * 100
data.frame(tot=nTot, totRcpp = nRcpp, totCompiled = nCompiled,
           RcppPctOfCompiled = propRcpp)
```

```
##     tot totRcpp totCompiled RcppPctOfCompiled
## 1 19144    2631        4470           58.8591
```

How? Review of How Things were

Simple Example

R Version of 'is this number odd or even'

```
isOdd_r <- function(num = 10L) {
   result = (num %% 2L == 1L)
   return(result)
}
isOdd_r(42L)
```

```
## [1] FALSE
```

Simple Example

R Version of 'is this number odd or even'

```
isOdd_r <- function(num = 10L) {
   result = (num %% 2L == 1L)
   return(result)
}
isOdd_r(c(42L, 43L, 44L))  # an aside: R automagically vectorised
```

```
## [1] FALSE  TRUE FALSE
```

Simple Example (cont.)

C++ Version of ‘is this number odd or even’

```cpp
bool isOdd_cpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}
```

Free-standing code, not yet executable, may need `Makefile`, ...

Simple Example (cont.)

Code for 'is_odd_prog.cpp' Program

```cpp
#include <iostream>

bool isOdd_cpp(int num = 10) {
   bool result = (num % 2 == 1);
   return result;
}

int main() {
   std::cout << "42: " << isOdd_cpp(42) << std::endl;
   std::cout << "43: " << isOdd_cpp(43) << std::endl;
   exit(0);
}
```

Compile and Run

```
$ g++ -o is_odd is_odd_prog.cpp
$ ./is_odd
42: 0
43: 1
$
```

Simple Example (cont.)

Code for `is_odd_R.cpp`

```cpp
#include <R.h>
#include <Rdefines.h>
#include <iostream>

bool isOdd_cpp(int num = 10) {
   bool result = (num % 2 == 1);
   return result;
}

extern "C" SEXP isOdd_call(SEXP numsx) {
   int num = Rf_asInteger(numsx);
   bool res = isOdd_cpp(num);
   SEXP resxp = Rf_ScalarInteger(res);
   return resxp;
}
```

Compiling and Linking

```
$ R CMD COMPILE  is_odd_R.cpp
$ R CMD SHLIB  is_odd_R.cpp
```

Loading and Running in R

```
dyn.load("is_odd_R.so")
.Call("isOdd_call", 42L)
.Call("isOdd_call", 43L)
```

Simple Example (cont.)

How was that?

- Extra legwork:
  - to get one integer "in"
  - and one bool / int out
- Rather manual steps of compiling, linking, loading
- Also operating system dependent: .so for me, .dll on Windows
- Call in R somewhat awkward via .Call()

Not great. But we have something better!

Simple Example (cont.)

Rcpp Version of 'is this number odd or even'

```
Rcpp::cppFunction("
bool isOdd_cpp(int num = 10) {
   bool result = (num % 2 == 1);
   return result;
}")
isOdd_cpp(42L)
```

```
## [1] FALSE
```

Simple Example (cont.)

In R

```
##
isOdd_r <- function(n=10L) {
   res = (n %% 2L == 1L)
   return(res)
}
isOdd_r(42L)
```

```
## [1] FALSE
```

In C++ via Rcpp

```
Rcpp::cppFunction("
bool isOdd_cpp(int n=10) {
   bool res = (n % 2 == 1);
   return res;
}")
isOdd_cpp(42L)
```

```
## [1] FALSE
```

How? The Rcpp Way

A Quick Preliminary Test

```
## evaluate simple expression
## ... as C++ code
Rcpp::evalCpp("2 + 2")

## [1] 4

## more complex example
set.seed(42)
Rcpp::evalCpp("Rcpp::rnorm(2)")

## [1]  1.370958 -0.564698
```

These steps should just work.

Windows users may need Rtools. macOS users need specific steps. Everybody else should have a compiler.

Consider *e.g.* `https://rstudio.cloud` for a working setup (albeit with limited free hours).

We will discuss the commands on the left in more detail in a bit.

Basic Usage: evalCpp()

As seen, `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] 1.79769e+308
```

First Exercise

Exercise 1

Evaluate an expression in C++ via `Rcpp::evalCpp()`

Basic Usage: cppFunction()

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

```
cppFunction("
    int exampleCpp11() {
        auto x = 10;          // guesses type
        return x;
}", plugins=c("cpp11"))     ## turns on C++11

## R function with same name as C++ function
exampleCpp11()
```

Second Exercise

```
library(Rcpp)
cppFunction("int f(int a, int b) { return(a + b); }")
f(21, 21)
```

Exercise 2

Write a C++ function on the R command-line via `cppFunction()`

Should the above work? Yes? No?

What can you see examining it?

Can you "break it" ?

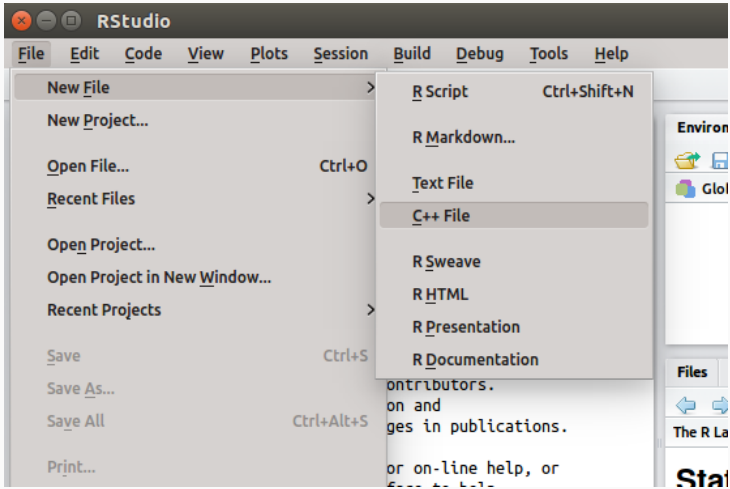
Basic Usage: sourceCpp()

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail in the package vignette Rcpp-attributes.

`sourceCpp()` builds on and extends `cxxfunction()` from package inline, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf RcppArmadillo, RcppEigen, RcppGSL). Plugins can also turn on support for C++11/C++14/C++17/C++20, OpenMP, and more.

Jumping Right In: Via RStudio



A First Example: Cont'ed

```cpp
#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 ask 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

Third Exercise

Exercise 3

Modify the `timesTwo` function used via `Rcpp::sourceCpp()`

Use the RStudio File -> New File -> C++ File template.

First Example: Speed

Consider a function defined as

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

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
```

Timing:

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

```
##    test replications elapsed relative
## 1 f(10)          100   0.008      1.0
## 2 f(15)          100   0.100     12.5
## 3 f(20)          100   1.116    139.5
```

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

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

Timing:

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

```
##    test replications elapsed relative
## 1 f(20)          100   1.186      593
## 2 g(20)          100   0.002        1
```

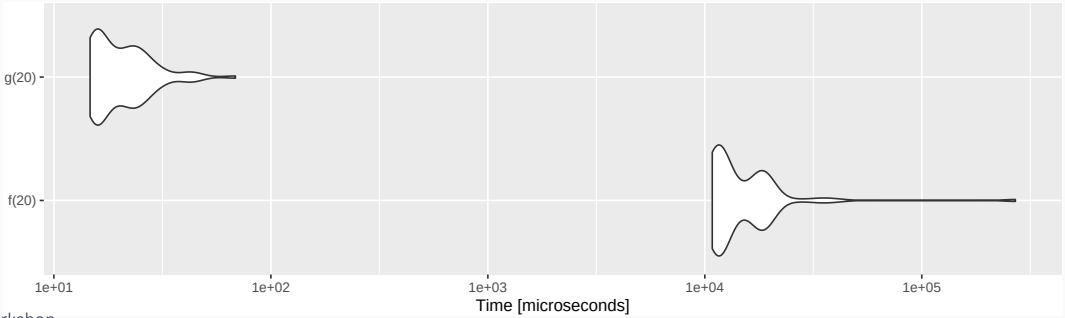
A nice gain of a few orders of magnitude.

An Introductory Example: Comparing timing

```
res <- microbenchmark::microbenchmark(f(20), g(20))
res
```

```
## Unit: microseconds
##   expr       min        lq      mean    median        uq        max neval cld
##  f(20) 10824.131 11454.662 17386.459 12066.750 18308.858 270105.723   100   b
##  g(20)    14.682    15.622    21.622    18.925    24.421     68.644   100  a
```

```
suppressMessages(microbenchmark:::autoplot.microbenchmark(res))
```



Fourth Exercise

```
// [[Rcpp::export]]
int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2));
}
```

Exercise 4

Run the C++ fibonacci function and maybe try some larger values.

Easiest:

- Add function to C++ file template
- Remember to add `// [[Rcpp::export]]`

A (Very) Brief C++ Primer

C++ is 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
```

Locations may be OS and/or installation-dependent

C++ is Statically Typed

Examples

- 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

Examples

- 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 ...

C++ is Object-Oriented

A 2nd 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;
};
```

A `struct` is (very loosely) a bit like an R `list`, contains no code, but can be nested.

C++ is Object-Oriented

```
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();
}
```

Classes extend structs with code

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

Here the `year`, `month`, and `date` integers are declared as 'private' variables and are accessed via 'getter' and 'setter' functions. (There is *much* more too all of this, of course.)

R Type mapping by Rcpp

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works.

A key feature of Rcpp: works with scalar types, R vectors, STL vectors.

C++ and R Types

So-called *atomistic* base types in C and C++ contain *one* value.

By contrast, in R everything is a *vector* so we have vector classes (as well as corresponding `*Matrix` classes like `NumericalMatrix`.

Basic C and C++: Scalar

- `int`
- `double`
- `char[]`; `std::string`
- `bool`
- `complex`

Rcpp Vectors

- `IntegerVector`
- `NumericVector`
- `CharacterVector`
- `LogicalVector`
- `ComplexVector`

Second Example: Vectors

Types: Vector Example

A “teaching-only” first example – there are better ways:

```
#include <Rcpp.h>
// [[Rcpp::export]]
double getMax(Rcpp::NumericVector v) {
    int n = v.size();  // vectors are self-describing, we can ask about size
    double m = v[0];   // initialize
    for (int i=0; i<n; i++) {
        if (v[i] > m) {
            Rcpp::Rcout << "Now " << m << std::endl;
            m = v[i];
        }
    }
    return(m);
}
```

Types: Vector Example

```
cppFunction("double getMax(NumericVector v) {
    int n = v.size();    // vectors are self-describing
    double m = v[0];     // initialize
    for (int i=0; i<n; i++) {
        if (v[i] > m) {
            m = v[i];
            Rcpp::Rcout << \"Now \" << m << std::endl;
        }
    }
    return(m);
}")
getMax(c(4,5,2))


## Now 5

## [1] 5
```

Another Vector Example: Column Sums

```
#include <Rcpp.h>

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

Another Vector Example: Column Sums

Key Elements

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

Exercise 5

Modify this vector example to compute on vectors

Compute a `min`. Or the `sum`. Or loop backwards.

Try a few things.

```
// [[Rcpp::export]]
double getMax(NumericVector v) {
    int n = v.size();  // vectors are describing
    double m = v[0];   // initialize
    for (int i=0; i<n; i++) {
        if v[i] > m {
            Rcpp::Rcout << "Now "
                        << m << std::endl;
            m = v[i];
        }
    }
    return(m);
}
```

Templated vectors

C++ has vectors as well: written as `std::vector<T>` where the T denotes 'template' meaning different *types* can be used to instantiate.

The key part is that Rcpp allows easy interoperation with them so we can work with countless C++ libraries.

```
cppFunction("double getMax2(std::vector<double> v) {
    int n = v.size();    // vectors are describing
    double m = v[0];     // initialize
    for (int i=0; i<n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return(m);
}")
getMax2(c(4,5,2))
```

```
## [1] 5
```

STL Vectors

Useful to know

- STL vectors are widely used so Rcpp supports them
- Very useful to access other C++ code and libraries
- One caveat: Rcpp vectors *reuse* R memory so no copies
- STL vectors have different underpinning so copies
- But not a concern unless you have
    - either HUGE data structurs,
    - or many many calls

One Important Issue

```
cppFunction("void setSecond(Rcpp::NumericVector v) {
    v[1] = 42;
}")
v <- c(1,2,3);    setSecond(v); v   # as expected
```

```
## [1]  1 42  3
```

```
v <- c(1L,2L,3L); setSecond(v); v   # different
```

```
## [1] 1 2 3
```

Exercise 6

Please reason about the previous example.

What might cause this?

Two More Things on Rcpp Vectors

Easiest solution on the `getMax()` problem:

```
double getMax(NumericVector v) {
    return( max( v ) );
}
```

Just use the *Sugar* function `max()`!

For Rcpp data structures we have *many* functions which act on C++ vectors just like their R counterparts.

But these may often often prefer Rcpp vectors over STL.

Vectors as data containers

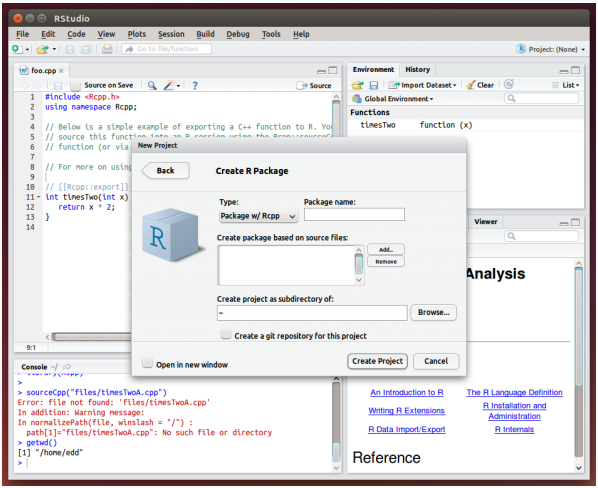
- Rcpp vectors (and matrices) do not really do linear algebra.
- In other words, do not use them for the usual “math” operations.
- Rather use RcppArmadillo – more on that later.

How: Packages

Packages

- *The* standard unit of R code organization.
- Creating packages with Rcpp is easy:
  - create an empty one via `Rcpp.package.skeleton()`
  - also `RcppArmadillo.package.skeleton()` for Armadillo
  - RStudio has the File -> New Project -> Package menu(s)
    (as we show on the next slide)
- The vignette Rcpp-packages has fuller details.
- As of February 2023, there are 2631 CRAN and 252 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

Packages and Rcpp



`Rcpp.package.skeleton()` and its derivatives.

e.g. `RcppArmadillo.package.skeleton()` create working packages.

```
// another simple example: outer product of a vector,
// returning a matrix
// [[Rcpp::export]]
arma::mat rcpparma_outerproduct(const arma::colvec & x) {
    arma::mat m = x * x.t();
    return m;
}

// and the inner product returns a scalar
// [[Rcpp::export]]
double rcpparma_innerproduct(const arma::colvec & x) {
    double v = arma::as_scalar(x.t() * x);
    return v;
}
```

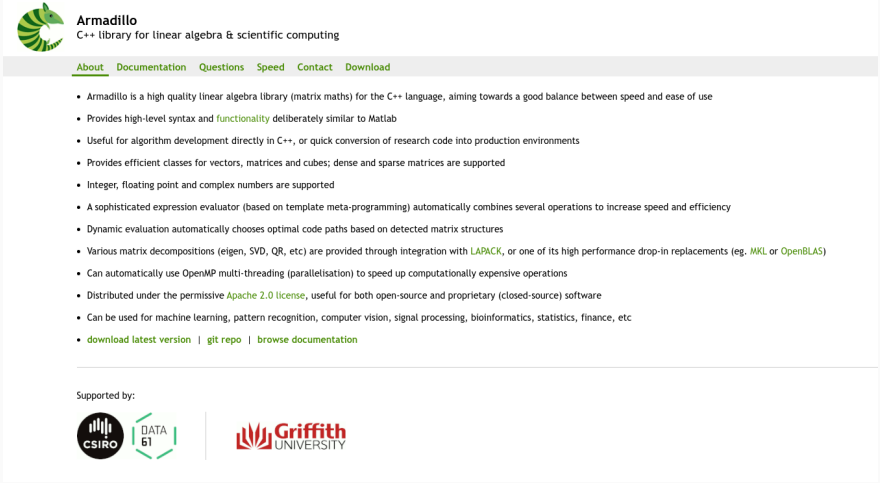
Two (or three) ways to link to external libraries

- *Full copies:* Do what RcppMLPACK (v1) does and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries:* Do what RcppGSL or RcppMLPACK (v2,v3) do and use hooks in the package startup to store compiler and linker flags which are passed to environment variables
- *With C++ template headers only:* Do what RcppArmadillo, mlpack (v4) and others do and just point to the headers

More details in extra vignettes. Not enough time here today to work through this.

RcppArmadillo

Armadillo



Source: http://arma.sf.net

What is Armadillo?

- Armadillo is a C++ linear algebra library (matrix maths) aiming towards a good balance between speed and ease of use.
- The syntax is deliberately similar to Matlab.
- Integer, floating point and complex numbers are supported.
- A delayed evaluation approach is employed (at compile-time) to combine several operations into one and reduce (or eliminate) the need for temporaries.
- Useful for conversion of research code into production environments, or if C++ has been decided as the language of choice, due to speed and/or integration capabilities.

Source: `http://arma.sf.net`

Key Points

- Provides integer, floating point and complex vectors, matrices, cubes and fields with all the common operations.
- Very good documentation and examples
  - website,
  - technical report (Sanderson, 2010),
  - CSDA paper (Sanderson and Eddelbuettel, 2014),
  - JOSS paper (Sanderson and Curtin, 2016),
  - ICMS paper (Sanderson and Curtin, 2018).
- Modern code, extending from earlier matrix libraries.
- Responsive and active maintainer, frequent updates.
- Used eg by MLPACK, see Curtin et al (JMLR 2013, JOSS 2023).

Key Points

- Template-only builds—no linking, and available whereever R and a compiler work (but Rcpp is needed)
- Easy to use, just add `LinkingTo: RcppArmadillo, Rcpp` to `DESCRIPTION` (*i.e.*, no added cost beyond Rcpp)
- Really easy from R via Rcpp and automatic converters
- Frequently updated, widely used – for example by now 1040 CRAN packages

Example: Column Sums

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::rowvec colSums(arma::mat mat) {
    size_t cols = mat.n_cols;
    arma::rowvec res(cols);

    for (size_t i=0; i<cols; i++) {
        res[i] = sum(mat.col(i));
    }
    return(res);
}
```

Example: Column Sums

Key Features

- The `[[Rcpp::depends(RcppArmadillo)]]` tag lets R tell `g++` (or `clang++`) about the need for Armadillo headers
- Dimension accessor via member variables `n_rows` and `n_cols`; not function calls
- We return a `rowvec`; default `vec` is alias for `colvec`
- Column accessor is just `col(i)` here
- This is a simple example of how similar features may have slightly different names across libraries

Example: Eigen values

```
#include <RcppArmadillo.h>

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

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

Example: Eigen values

```
Rcpp::sourceCpp("code/armaeigen.cpp")
M <- cbind(c(1,-1), c(-1,1))
getEigenValues(M)
```

```
##      [,1]
## [1,]    0
## [2,]    2
```

```
eigen(M)[["values"]]
```

```
## [1] 2 0
```

Exercise 7

Write an inner and outer product of a vector

Hints:

- `arma::mat` and `arma::colvec` (aka `arma::vec`) are useful
- the `.t()` function transposes
- `as_scalar()` lets you assign to a `double`

Vector products

```cpp
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// simple example: outer product of a vector, returning a matrix
//
// [[Rcpp::export]]
arma::mat rcpparma_outerproduct(const arma::colvec & x) {
    arma::mat m = x * x.t();
    return m;
}

// and the inner product returns a scalar
//
// [[Rcpp::export]]
double rcpparma_innerproduct(const arma::colvec & x) {
    double v = arma::as_scalar(x.t() * x);
    return v;
}
```

Straightforward

- The package itself contains the `RcppArmadillo.package.skeleton()` helper
- RStudio also offers File -> New Project -> (New | Existing) Directory -> Package with RcppArmadillo
- *Easy* and reliable to deploy as header-only without linking
- One caveat on macOS is the need for `gfortran`, see online help

Third Example: FastLm

Background

- Implementations of `fastLm()` have been a staple during the early development of Rcpp
- First version was in response to a question on r-help.
- Request was for a fast function to estimate parameters – and their standard errors – from a linear model,
- It used GSL functions to estimate $\hat{\beta}$ as well as its standard errors $\hat{\sigma}$ – as `lm.fit()` in R only returns the former.
- It has since been reimplemented for RcppArmadillo and RcppEigen

Initial FastLm

```cpp
#include <RcppArmadillo.h>

extern "C" SEXP fastLm(SEXP Xs, SEXP ys) {
    try {
        Rcpp::NumericVector yr(ys);                     // creates Rcpp vector from SEXP
        Rcpp::NumericMatrix Xr(Xs);                     // creates Rcpp matrix from SEXP
        int n = Xr.nrow(), k = Xr.ncol();
        arma::mat X(Xr.begin(), n, k, false);           // reuses memory, avoids extra copy
        arma::colvec y(yr.begin(), yr.size(), false);

        arma::colvec coef = arma::solve(X, y);          // fit model y ~ X
        arma::colvec res  = y - X*coef;                 // residuals, and 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 Rcpp::List::create(Rcpp::Named("coefficients") = coef,
                                  Rcpp::Named("stderr")       = std_err,
                                 Rcpp::Named("df.residual")  = n - k   );
    } catch( std::exception &ex ) {
        forward_exception_to_r( ex );
    } catch(...) {
        ::Rf_error( "c++ exception (unknown reason)" );
    }
    return R_NilValue; // -Wall
}
```

Newer Version

```
// [[Rcpp::depends(RcppArmadillo)]]
#include <RcppArmadillo.h>
using namespace Rcpp;
using namespace arma;

// [[Rcpp::export]]
List fastLm(NumericVector yr, NumericMatrix Xr) {
   int n = Xr.nrow(), k = Xr.ncol();
   mat X(Xr.begin(), n, k, false);
   colvec y(yr.begin(), yr.size(), false);

   colvec coef = solve(X, y);
   colvec resid = y - X*coef;

   double sig2 = as_scalar(trans(resid)*resid/(n-k));
   colvec stderrest = sqrt(sig2 * diagvec( inv(trans(X)*X)) );

   return List::create(Named("coefficients") = coef,
                       Named("stderr")       = stderrest,
                       Named("df.residual")  = n - k   );
}
```

Current Version

```
// [[Rcpp::depends(RcppArmadillo)]]
#include <RcppArmadillo.h>

// [[Rcpp::export]]
Rcpp::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);
   arma::colvec resid = y - X*coef;

   double sig2 = arma::as_scalar(arma::trans(resid)*resid/(n-k));
   arma::colvec sterr = arma::sqrt(sig2 * arma::diagvec(arma::pinv(arma::trans(X)*X)));

   return Rcpp::List::create(Rcpp::Named("coefficients") = coef,
                             Rcpp::Named("stderr")       = sterr,
                             Rcpp::Named("df.residual")  = n - k );
}
```

Interface Changes

```
arma::colvec y = Rcpp::as<arma::colvec>(ys);
arma::mat X = Rcpp::as<arma::mat>(Xs);
```

Convenient, but additional copy. Next variant uses two steps, but only pointer is copied:

```
Rcpp::NumericVector yr(ys);
Rcpp::NumericMatrix Xr(Xs);
int n = Xr.nrow(), k = Xr.ncol();
arma::mat X(Xr.begin(), n, k, false);
arma::colvec y(yr.begin(), yr.size(), false);
```

Better if performance is concern. And RcppArmadillo now has efficient `const references` implementing this for us 'behind the scenes'.

Benchmark

```
edd@brad:~$ Rscript ~/git/rcpparmadillo/inst/examples/fastLm.r
                      test replications relative elapsed
2         fLmTwoCasts(X, y)         5000    1.000   0.072
4             fLmSEXP(X, y)         5000    1.000   0.072
3         fLmConstRef(X, y)         5000    1.014   0.073
6   fastLmPureDotCall(X, y)         5000    1.028   0.074
1          fLmOneCast(X, y)         5000    1.250   0.090
5          fastLmPure(X, y)         5000    1.486   0.107
8              lm.fit(X, y)         5000    2.542   0.183
7 fastLm(frm, data = trees)         5000   36.153   2.603
9     lm(frm, data = trees)         5000   43.694   3.146
## continued below with subset
```

Benchmark

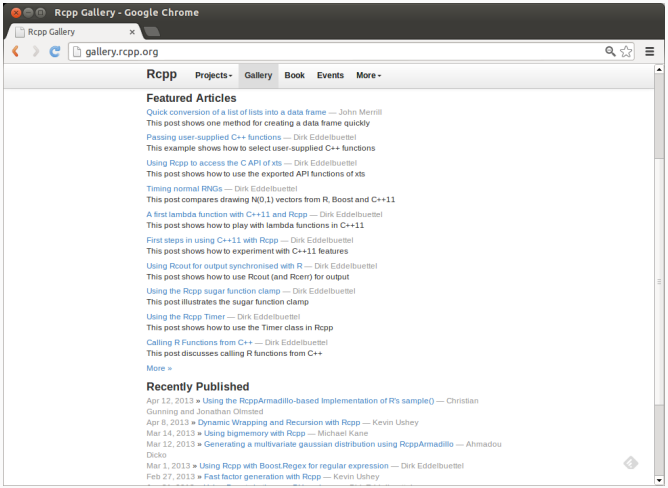
```
## continued from above with larger N
                     test replications relative elapsed
1        fLmOneCast(X, y)        50000    1.000   0.676
3           fLmSEXP(X, y)        50000    1.027   0.694
4       fLmConstRef(X, y)        50000    1.061   0.717
6 fastLmPureDotCall(X, y)        50000    1.061   0.717
2       fLmTwoCasts(X, y)        50000    1.123   0.759
5        fastLmPure(X, y)        50000    1.583   1.070
7            lm.fit(X, y)        50000    2.530   1.710
edd@brad:~$
```

More

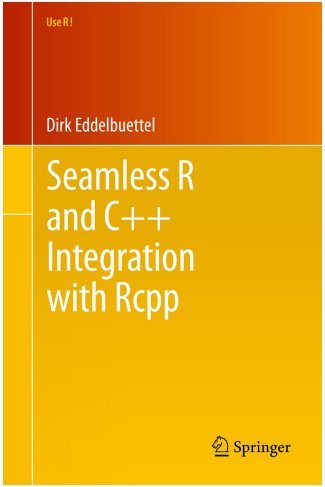
Where to go for next steps

- The package comes with nine pdf vignettes, and help pages.
- The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp Stat & Data Anlys*, Rcpp again in *TAS*)
- The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has nearly 3000 posts too – which can be searched.
- And a number of blog posts introduce/discuss features.

Rcpp Gallery



The Rcpp book



On sale since June 2013.

Thank you!

slides	https://dirk.eddelbuettel.com/presentations/
web	https://dirk.eddelbuettel.com/
mail	dirk@eddelbuettel.com
github	@eddelbuettel
twitter	@eddelbuettel
mastodon	@eddelbuettel@mastodon.social