



SEAMLESS R AND C++ INTEGRATION WITH RCPP

Dirk Eddebuettel

Online Tutorial for *useR!* 2020

14 July 2020

BEFORE WE GET STARTED

Special **Thank You!** to



- organizers of useR! 2020 in St. Louis
- organizers of iseR! 2020 satellite in Munich
- R Ladies Groups
 - Santiago
 - Valparaíso
 - Concepción
- R User Group Santiago

OVERVIEW

Overview

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

WHY R?



Why use the R Language?

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

Translations

Russian: <http://clpartmag.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 modeling, 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 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.

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



Why R for data analysis?

R is not the only language that can be used for data analysis. Why R rather than another? Here is a list:

- interactive language
- data structures
- graphics
- missing values
- functions as first class objects
- packages
- community

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

WHY R: PROGRAMMING WITH DATA

R as an 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



From any one of

- csv
- txt
- xlsx
- xml, json, ...
- web scraping, ...
- hdf5, netcdf, ...
- sas, stata, spss, ...
- various SQL + NOSQL DBs
- various binary protocols

via

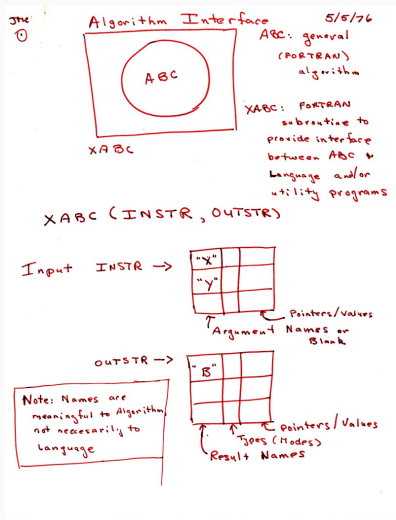


into any one of

- txt
- html
- latex and pdf
- html and js
- word
- shiny
- most graphics formats
- other dashboards
- web frontends

WHY R: HISTORICAL PERSPECTIVE

R AS 'THE INTERFACE'



A sketch of '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;
Chambers (2020), S, R, and Data Science, Proc ACM Prog
Lang, doi.org/10.1145/3386334.

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*.

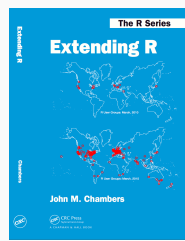
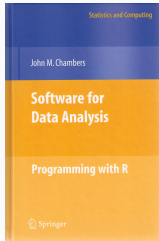
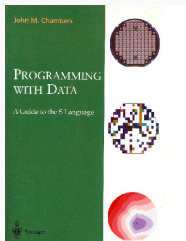
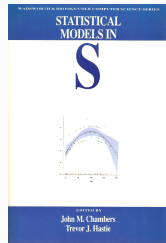
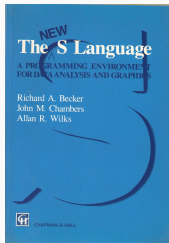
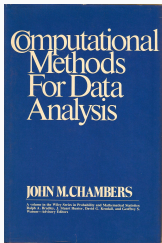
Additional Key Words and Phrases: data science, statistical computing, scientific computing

ACM Reference Format:

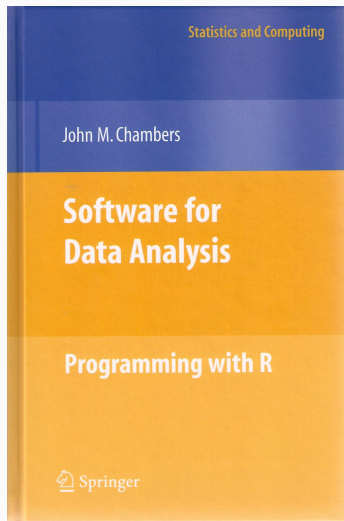
John M. Chambers. 2020. S, R, and Data Science. *Proc. ACM Program. Lang.* 4, HOPL, Article 84 (June 2020), 17 pages. <https://doi.org/10.1145/3386334>

Very recent, and
very nice, paper.

WHY R? : *PROGRAMMING WITH DATA* FROM 1977 TO 2016

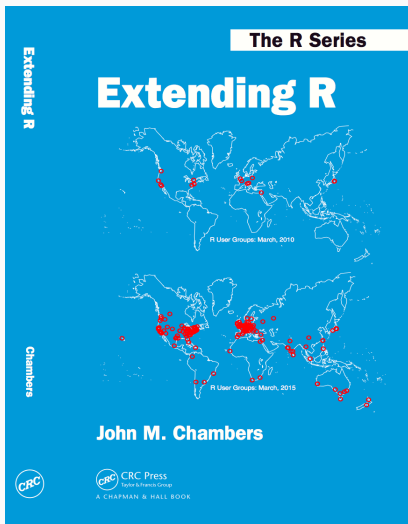


Thanks to John Chambers for high-resolution cover images. The publication years are, respectively, 1977, 1988, 1992, 1998, 2008 and 2016.



Software For Data Analysis

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

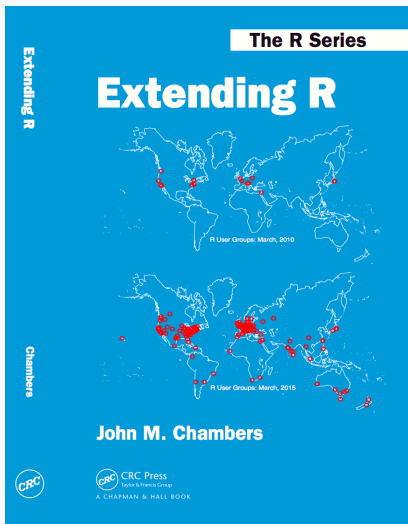


Extending R

Object: Everything that exists in R is an object

Function: Everything happens in R is a function call

Interface: Interfaces to other software are part of R



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 is likely to be ideal for all aspects. Interfacing multiple systems is the essence. Part IV explores the design 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 takes one or more of the high-level data structures R uses
- ... and returns one. Formally:

```
SEXP .Call(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**
 - the **SEXP** 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 that complicated – there are numerous examples
- Easy to use as it avoids build and OS system complexities thanks to the R infrastructure
- 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 directly or via eg *Rcpp modules*

Who ?

Growth of Rcpp usage on CRAN

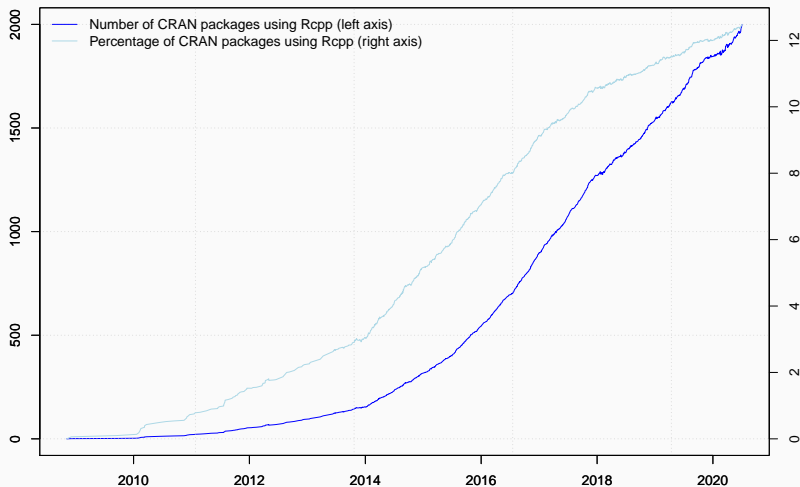


Figure reflects data as of 4 July 2020.

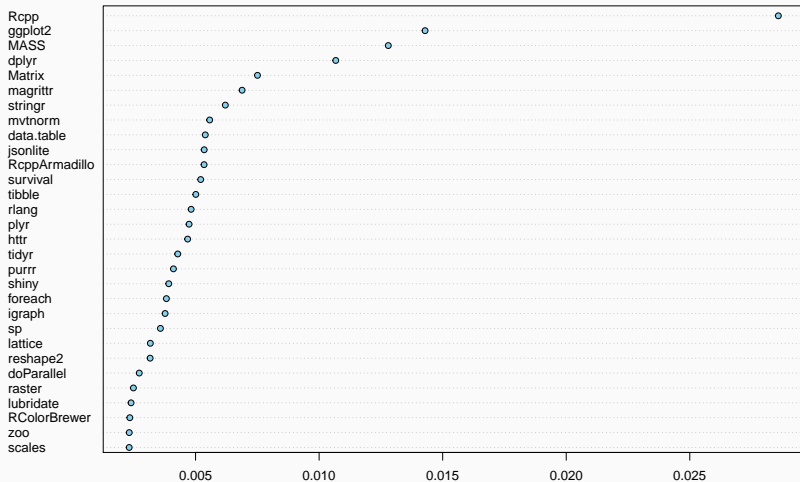
Rcpp is currently used by

- 2013 CRAN packages
- 203 BioConductor packages
- an unknown (“large”) number of GitHub, GitLab, ... projects

```
suppressMessages(library(utils))  
library(pagerank)    # cf github.com/andrie/pagerank  
  
cran <- "http://cloud.r-project.org"  
pr <- compute_pagerank(cran)  
round(100*pr[1:5], 3)
```

##	Rcpp	ggplot2	MASS	dplyr	Matrix
##	2.858	1.429	1.280	1.067	0.751

Top 30 of Page Rank as of July 2020



PERCENTAGE OF COMPILED PACKAGES

```
db <- tools::CRAN_package_db()    # added in R 3.4.0
db <- db[!duplicated(db[,1]),]
nTot <- nrow(db)
## all direct Rcpp reverse depends, ie packages using Rcpp
nRcpp <- length(tools::dependsOnPkgs("Rcpp", recursive=FALSE,
                                   installed=db))
nComp <- table(db[, "NeedsCompilation"])[["yes"]]
print(data.frame(tot=nTot, totRcpp=nRcpp, pctTot=nRcpp/nTot*100,
                 totComp=nComp, pctComp=nRcpp/nComp*100),
       row.names=FALSE)
```

```
##      tot totRcpp  pctTot totComp pctComp
## 16050    2013 12.5421   3916 51.4045
```

Calculations updated on 13 July 2020.

How?

A QUICK PRELIMINARY TEST

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

## [1] 4

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

## [1] 1.370958 -0.564698
```

This should just work.

Windows users may need Rtools.
Everybody else has a compiler.

Use <https://rstudio.cloud>
for a working setup.

We discuss the commands on
the left in a bit.

HOW: MAIN TOOLS FOR EXPLORATION

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
```

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 and examine it?

Can you “break it” ?

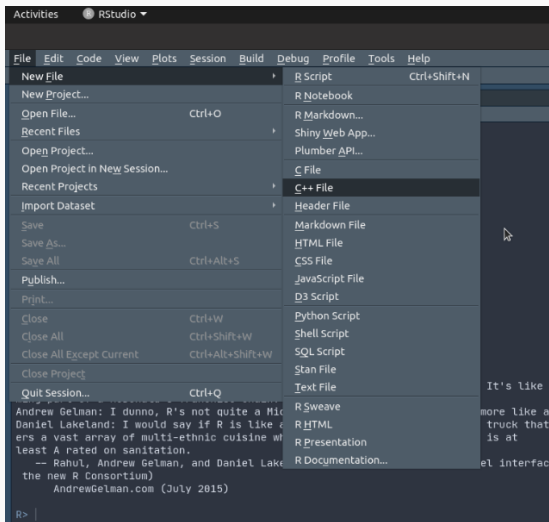
BASIC USAGE: SOURCECPP()

`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, OpenMP, and more.

JUMPING RIGHT IN: VIA RSTUDIO



A FIRST EXAMPLE: CONT'ED

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

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). ...

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
    return x * 2;
}

// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.

/**** R
timesTwo(42)
*/
```

So what just happened?

- We defined a simple C++ function
- It operates on a numeric vector argument
- We 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

Exercise 3

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

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

FIRST EXAMPLE: SPEED

AN EXAMPLE WITH FOCUS ON SPEED

Consider a function defined as

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

AN INTRODUCTORY EXAMPLE: 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	0.009	1.000
## 2	f(15)	100	0.094	10.444
## 3	f(20)	100	1.017	113.000

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(20), g(20))[,1:4]
```

##	test	replications	elapsed	relative
## 1	f(20)	100	1.011	505.5
## 2	g(20)	100	0.002	1.0

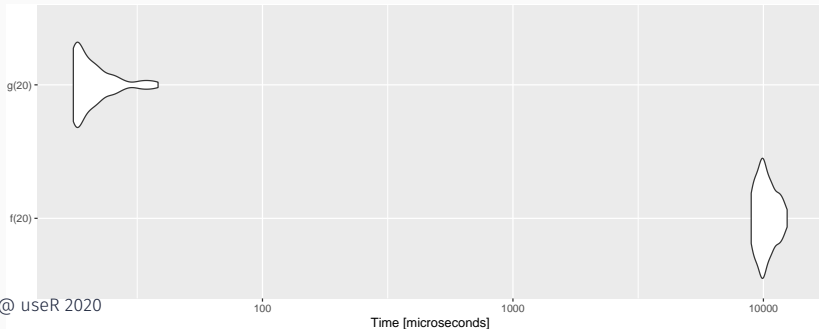
A nice gain of a few orders of magnitude.

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) 8938.83 9614.76 10257.36 10026.38 10806.23 12439.03   100   b  
##  g(20)  17.57  18.09   21.38   19.67   22.43   38.29   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

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 R: `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;  
};
```


C++ IS 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();  
}
```

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.

So-called *atomic* 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 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);
}
```

TYPES: VECTOR EXAMPLE

```
cppFunction("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) {  
      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

FIFTH EXERCISE

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);
}
```

STL VECTORS

C++ has vectors as well: written as `std::vector<T>` where the `T` denotes template meaning different *types* can be used to instantiate.

```
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
```

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 structures,
 - 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 structure we have *many* functions which act on C++ vectors just like their R counterparts.

But this requires Rcpp vectors – not STL.

Little explicit Math support

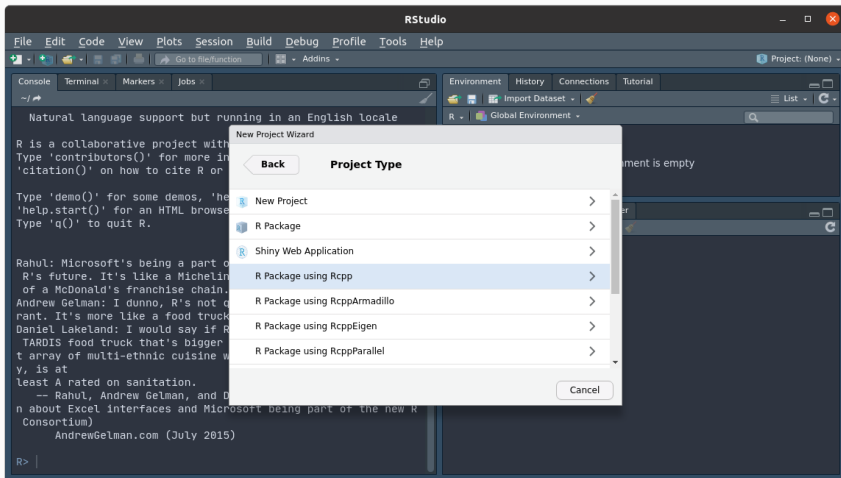
- Rcpp vectors (and matrices) do not really do linear algebra.
- So do not use them for the usual “math” operations.
- Rather use RcppArmadillo – more on that later.

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 -> Directory Choices -> Package Choices (as we show on the next slide)
- The vignette [Rcpp-packages](#) has fuller details.
- As of July 2020, there are about 2013 CRAN and 203 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

PACKAGES AND RCPP



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) 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 and other do and just point to the headers

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

RCPPARMADILLO



Armadillo

C++ library for linear algebra & scientific computing

[About](#) [Questions](#) [License](#) [Documentation](#) [Speed](#) [Contact](#) [Download](#)

- Armadillo is a high quality linear algebra library (matrix maths) for the C++ language, aiming towards a good balance between speed and ease of use
- Provides high-level syntax and [functionality](#) deliberately similar to Matlab
- Useful for algorithm development directly in C++, or quick conversion of research code into production environments (eg. software & hardware products)
- Provides efficient classes for vectors, matrices and cubes (1st, 2nd and 3rd order tensors); dense and sparse matrices are supported
- Integer, floating point and complex numbers are supported
- Various matrix decompositions are provided through integration with [LAPACK](#), or one of its high performance drop-in replacements (eg. multi-threaded [Intel MKL](#), or [OpenBLAS](#))
- A sophisticated expression evaluator (based on template meta-programming) automatically combines several operations to increase speed and efficiency
- Can automatically use OpenMP multi-threading (parallelisation) to speed up computationally expensive operations
- Available under a [permissive license](#), useful for both open-source and proprietary (closed-source) software
- Can be used for machine learning, pattern recognition, computer vision, signal processing, bioinformatics, statistics, finance, etc
- [download latest version](#) | [GitLab repo](#) | [browse documentation](#)

Supported by:



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 2018](#)).

Key Points

- Template-only builds—no linking, and available wherever 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 – now 756 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);
}
```

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/arma_eigenvalues.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

```
#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) Direction -> Package with RcppArmadillo
- *Easy* and reliable to deploy as header-only without linking
- 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 been reimplemented for RcppArmadillo and RcppEigen

INITIAL FASTLM

```
#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  $y \sim X$ 
    arma::colvec res = y - X*coef;         // residuals
    double s2 = std::inner_product(res.begin(), res.end(), res.begin(), 0.0)/(n - k);
    arma::colvec std_err =                 // std.errors of coefficients
      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
}
```

```
// [[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, yet incurs an additional copy. Next variant uses two steps, but only a pointer to objects 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 a concern. But now RcppArmadillo has efficient **const references** too.

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
```

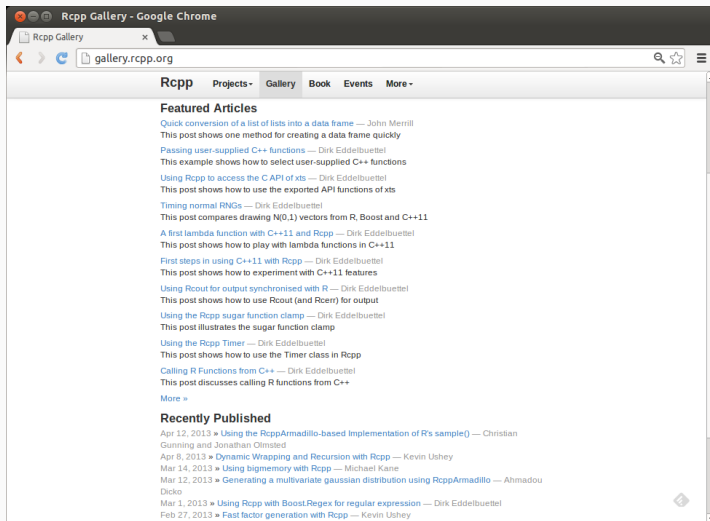
		test	replications	relative	elapsed
1	fLmOneCast(X, y)	50000	1.000	0.676	
3	fLmSEXP(X, y)	50000	1.027	0.694	
4	fLmConstRef(X, y)	50000	1.061	0.717	
6	fastLmPureDotCall(X, y)	50000	1.061	0.717	
2	fLmTwoCasts(X, y)	50000	1.123	0.759	
5	fastLmPure(X, y)	50000	1.583	1.070	
7	lm.fit(X, y)	50000	2.530	1.710	

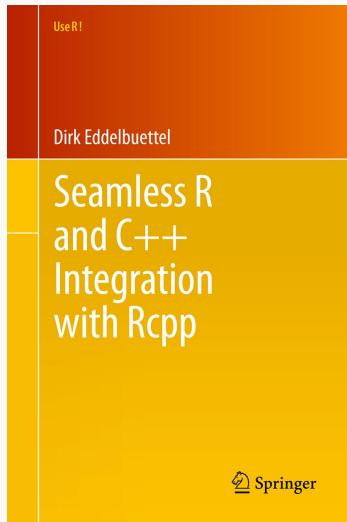
```
edda@brad:~$
```


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





On sale since June 2013.

THANK YOU!

slides <http://dirk.eddelbuettel.com/presentations/>

web <http://dirk.eddelbuettel.com/>

mail dirk@eddelbuettel.com

github [@eddelbuettel](#)

twitter [@eddelbuettel](#)