ILLINOIS

Introduction to Rcpp

From Simple Examples to Machine Learning

Dirk Eddelbuettel

useR! 2019 Pre-Conference Tutorial

Toulouse, France

9 July 2019

Overview

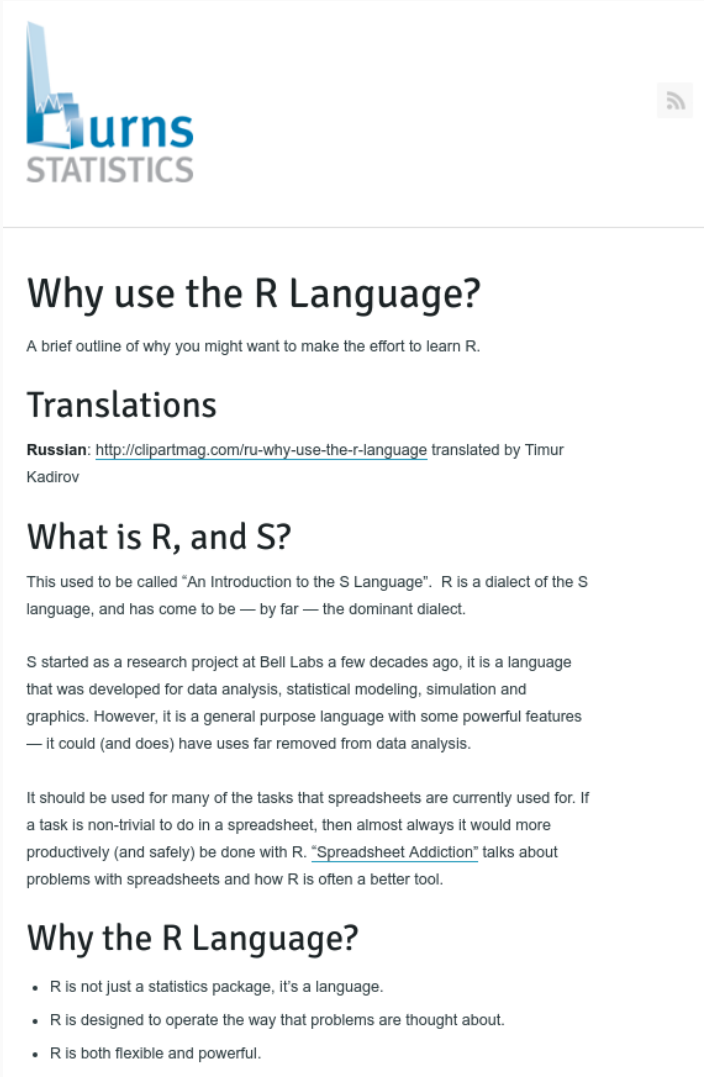
Very Broad Outline

Overview

- Motivation: Why R?
- Who Use This?
- How Does 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 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 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.

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

Why R? Pat Burn's View

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

Why R? R as Middle Man

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

R as central point

R as central point

From any one of

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

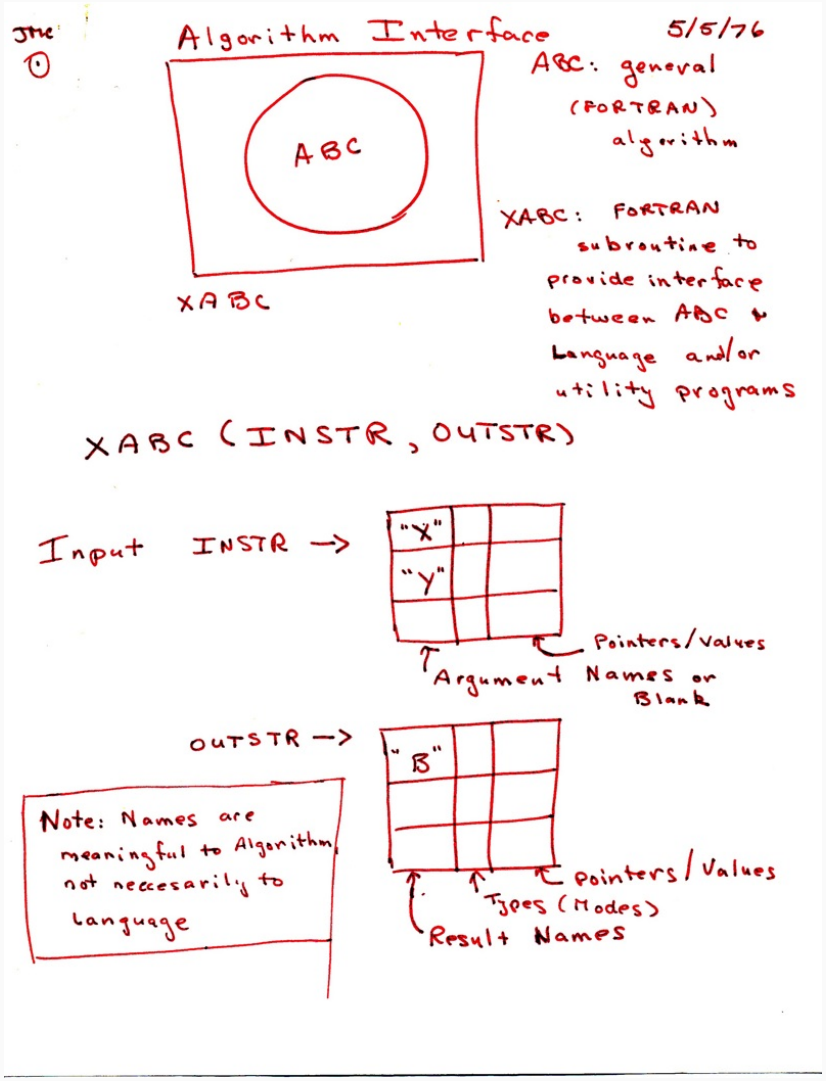
via

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 design sketch called 'The Interface'

AT&T Research lab meeting notes

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

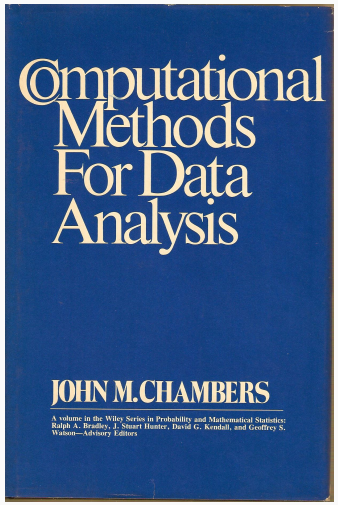
Key idea of abstracting away inner details giving higher-level more accessible view for user / analyst

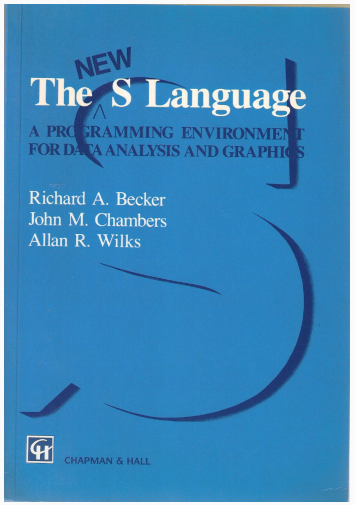
Lead to "The Interface"

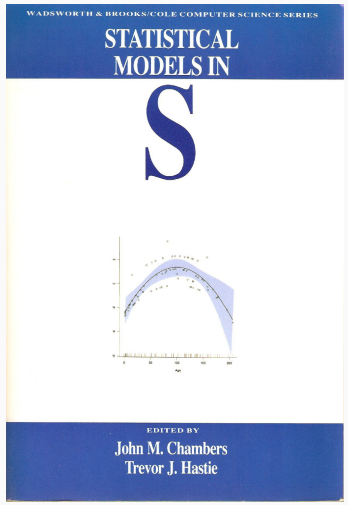
Which became S which lead to R

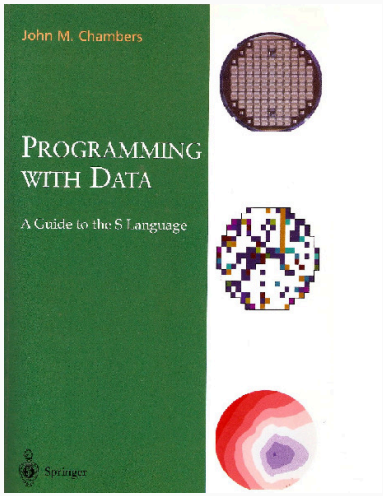
Source: John Chambers, personal communication

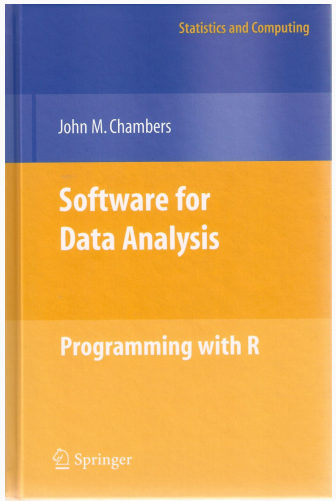
Why R? : *Programming with Data* from 1977 to 2016

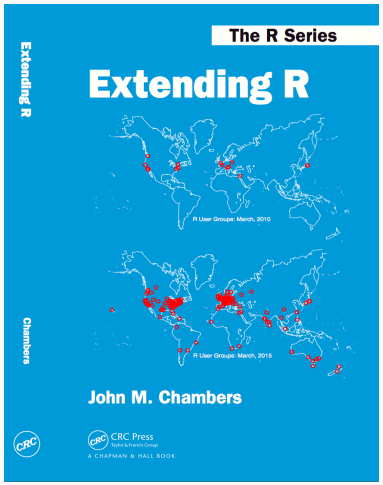






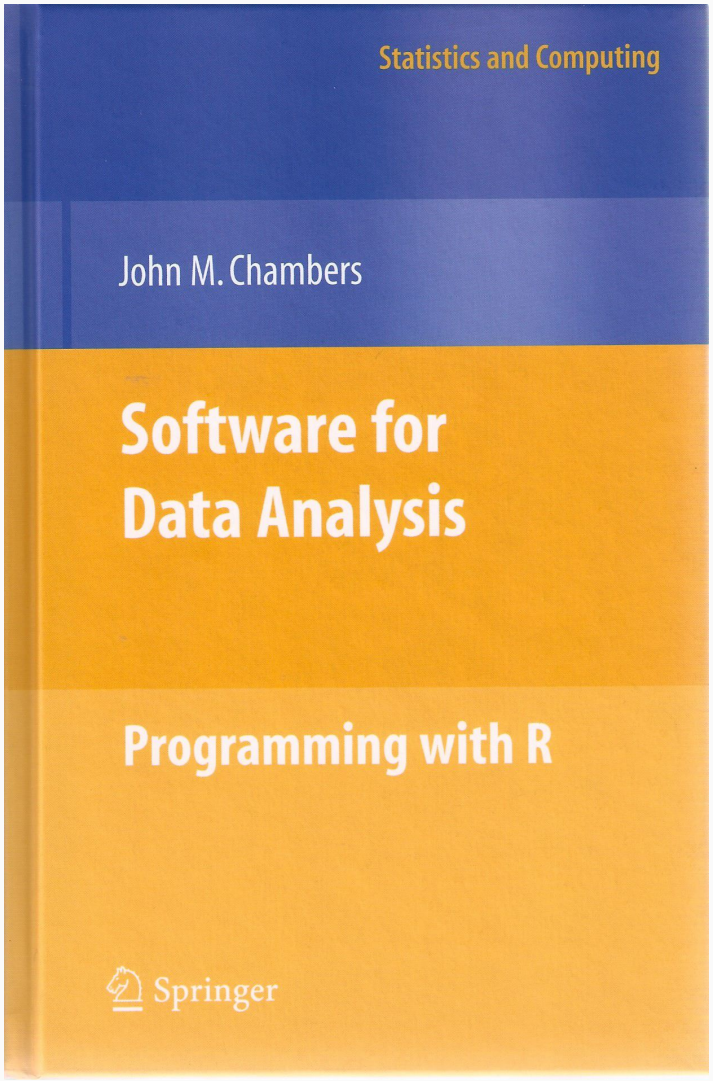






Thanks to John Chambers for high-resolution cover images. The publication years are, respectively, 1977, 1988, 1992, 1998, 2008 and 2016.

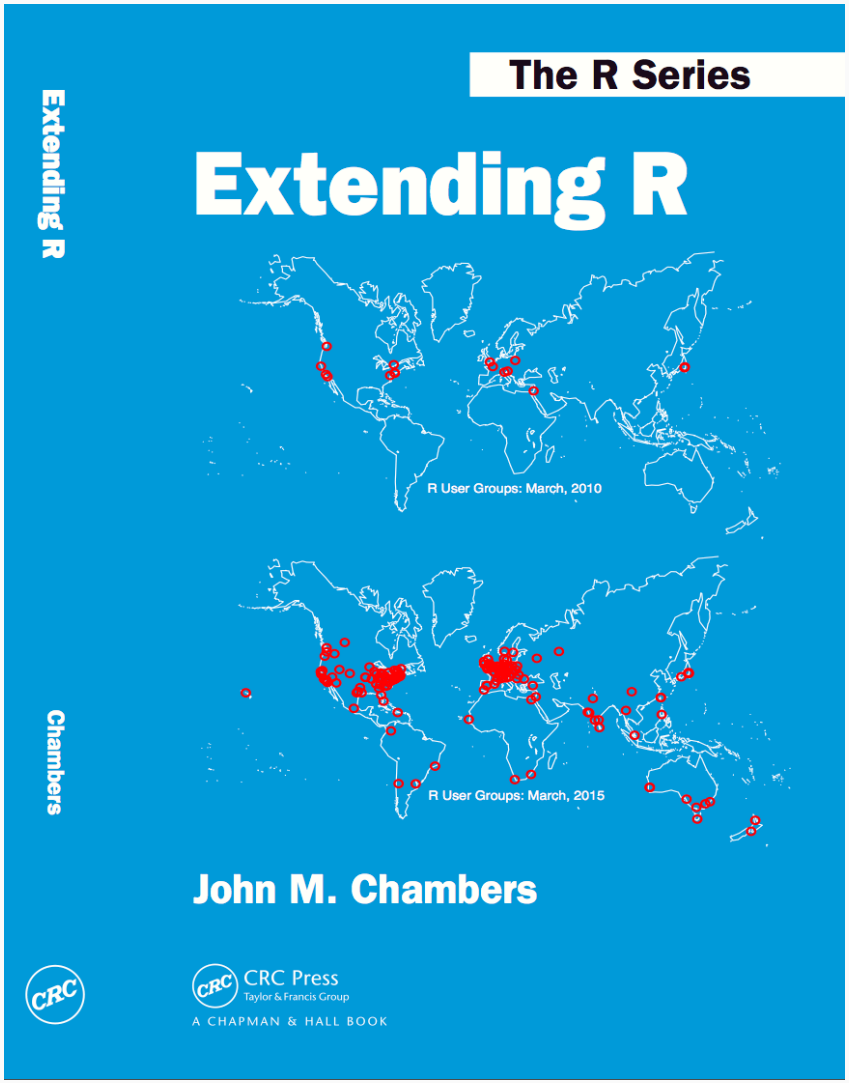
Chambers (2008)



Software For Data Analysis

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

Chambers (2016)



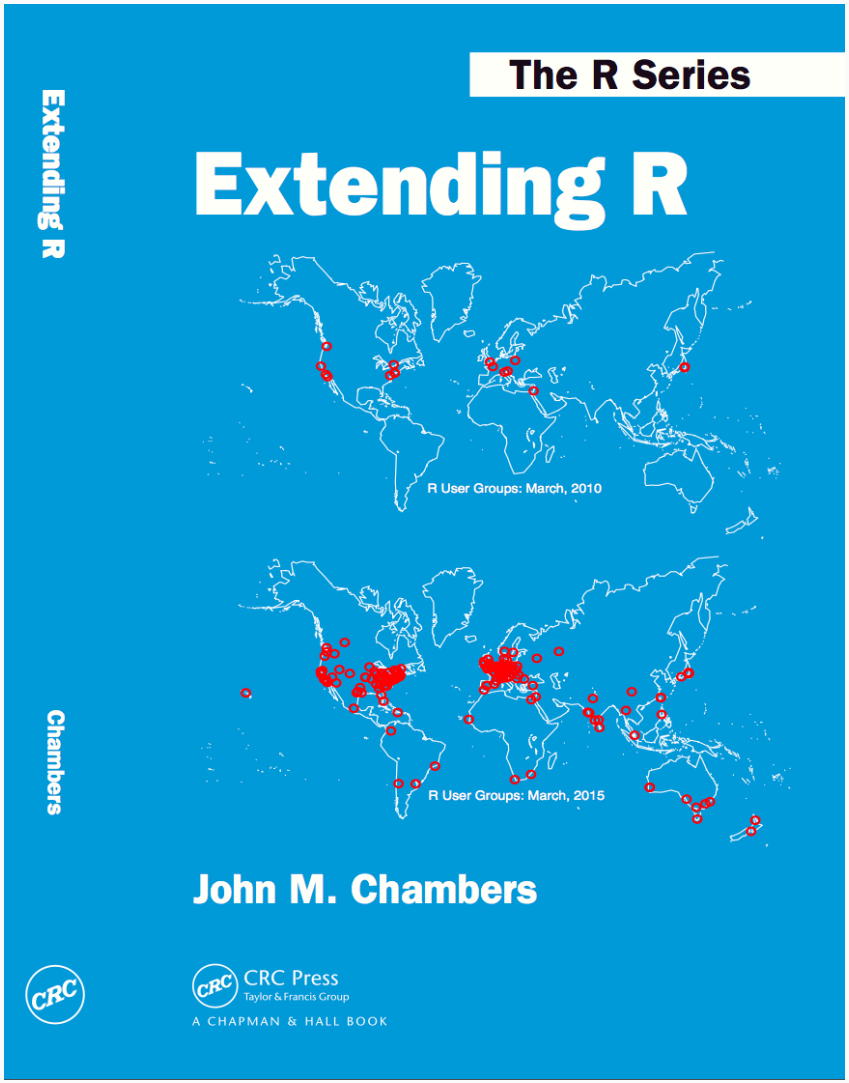
Extending R

Object: Everything that exists in R is an object

Function: Everything 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*

R and C/C++

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

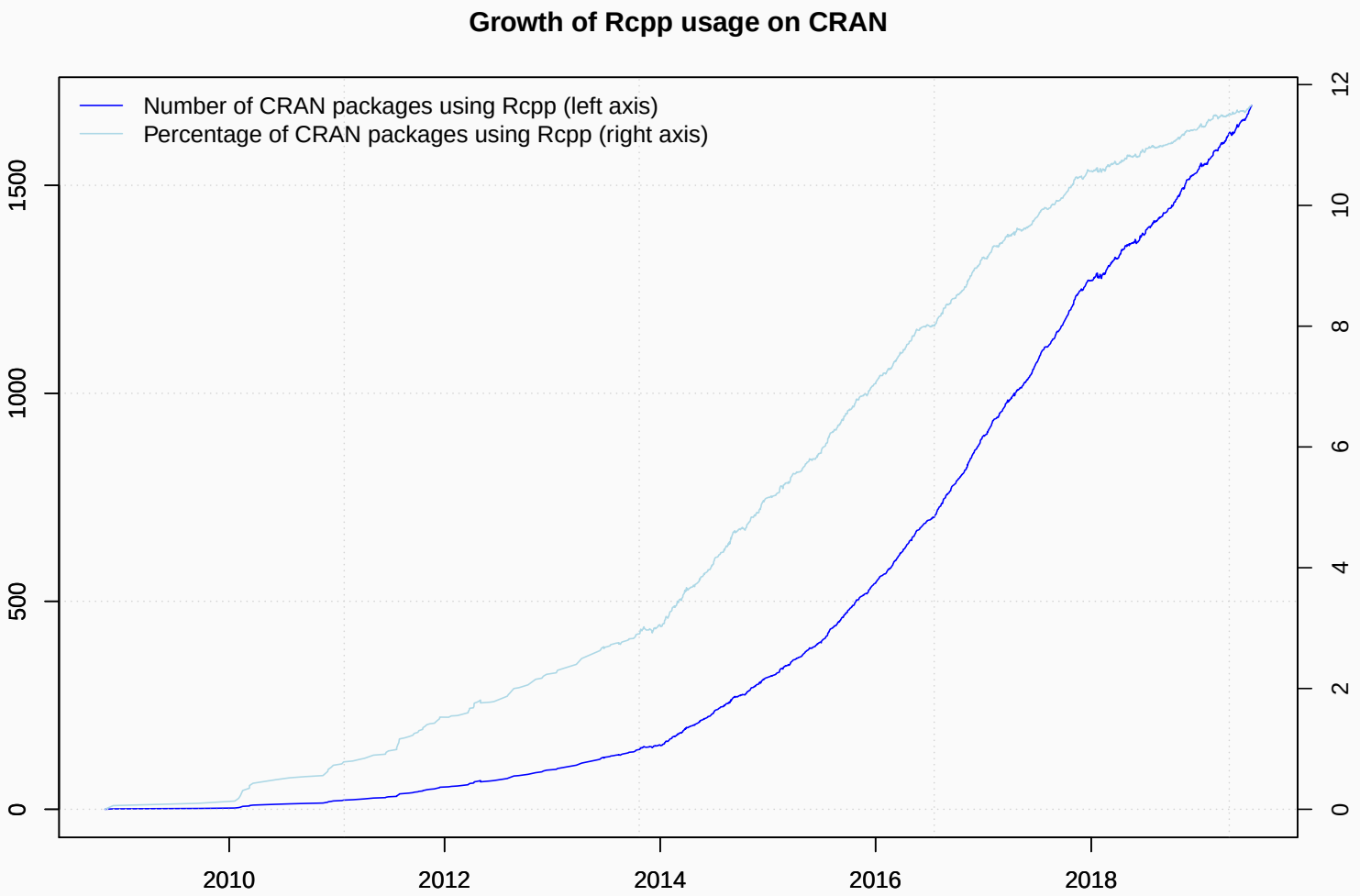
- It is *fast* – compiled C++ is hard to beat in other languages
  - (That said, you can *of course* write bad and slow code....)
- It is *very general* and widely used
  - *many libraries*
  - many tools
- It is fairly universal:
  - just about anything will have C interface so C++ can play
  - just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

Who ?

Growth



Data current as of June 29, 2019.

Users on Core Repositories

Rcpp is currently used by

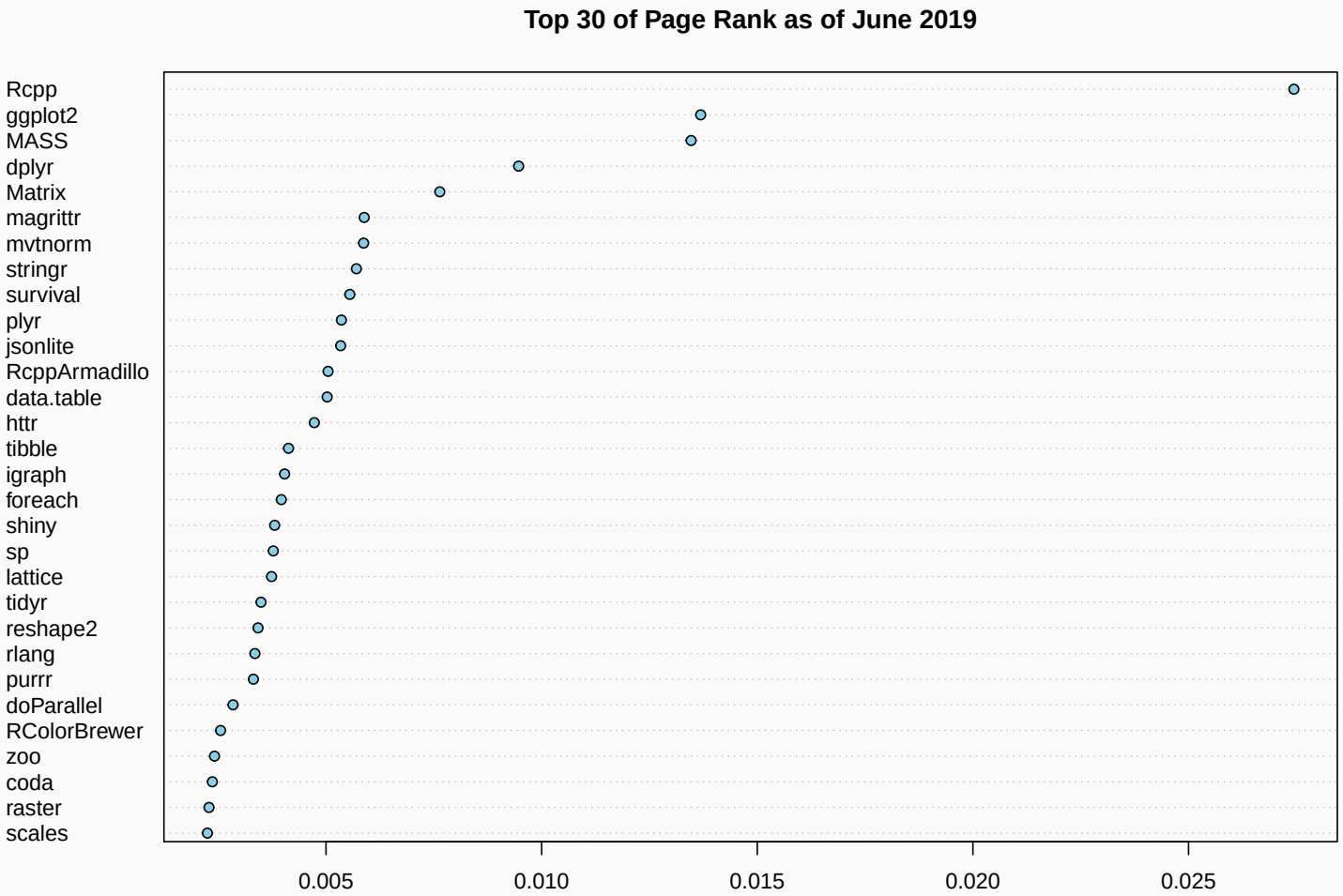
- 1692 CRAN packages (with 297 added since last year)
- 176 BioConductor packages (with 38 added since last year)
- an unknown (but “large”) number of GitHub projects

Pagerank

```
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
##   2.744   1.369   1.347   0.947   0.764
```

Pagerank


Top 30 of Page Rank as of June 2019
Rcpp
ggplot2
MASS
dplyr
Matrix
magrittr
mvtnorm
stringr
survival
plyr
jsonlite
RcppArmadillo
data.table
httr
tibble
igraph
foreach
shiny
sp
lattice
tidyr
reshape2
rlang
purrr
doParallel
RColorBrewer
zoo
coda
raster
scales
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
## rows: number of pkgs, cols: different attributes
nTot <- nrow(db)
## all direct Rcpp reverse depends, ie packages using Rcpp
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 14522    1686        3635          46.38239
```

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]  1.3709584 -0.5646982
```

This should just work.

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

Use `http://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] 1.797693e+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()`

Can you "break it"?

What can you see examining it?

Should the above work? Yes? No?

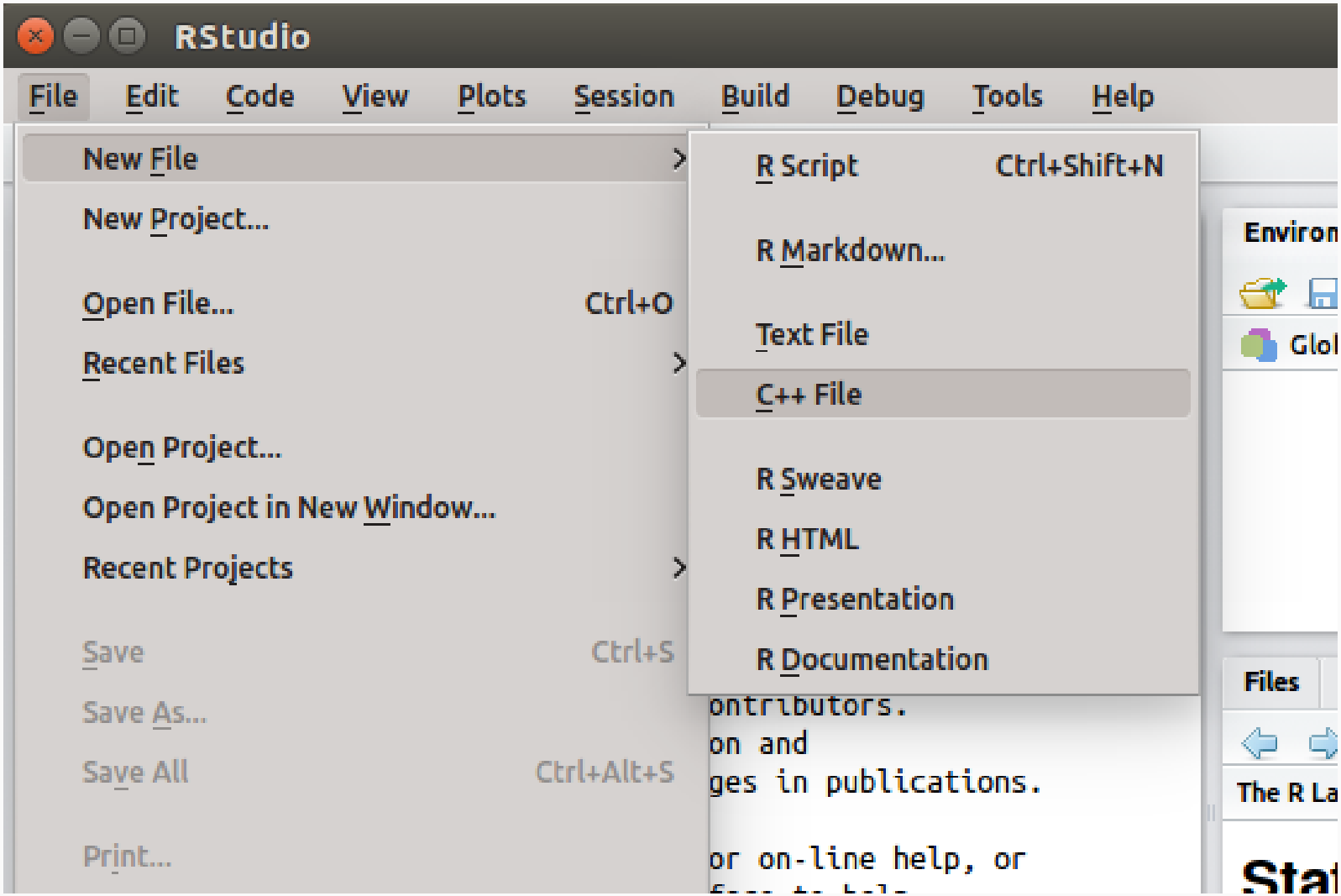
Basic Usage: sourceCpp()

`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)
*/
```

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

An Example with Focus on Speed

Consider a function defined as

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

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   0.014    1.000
## 2 f(15)          100   0.142   10.143
## 3 f(20)          100   1.565  111.786
```

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   1.581    263.5
## 2 g(20)          100   0.006      1.0
```

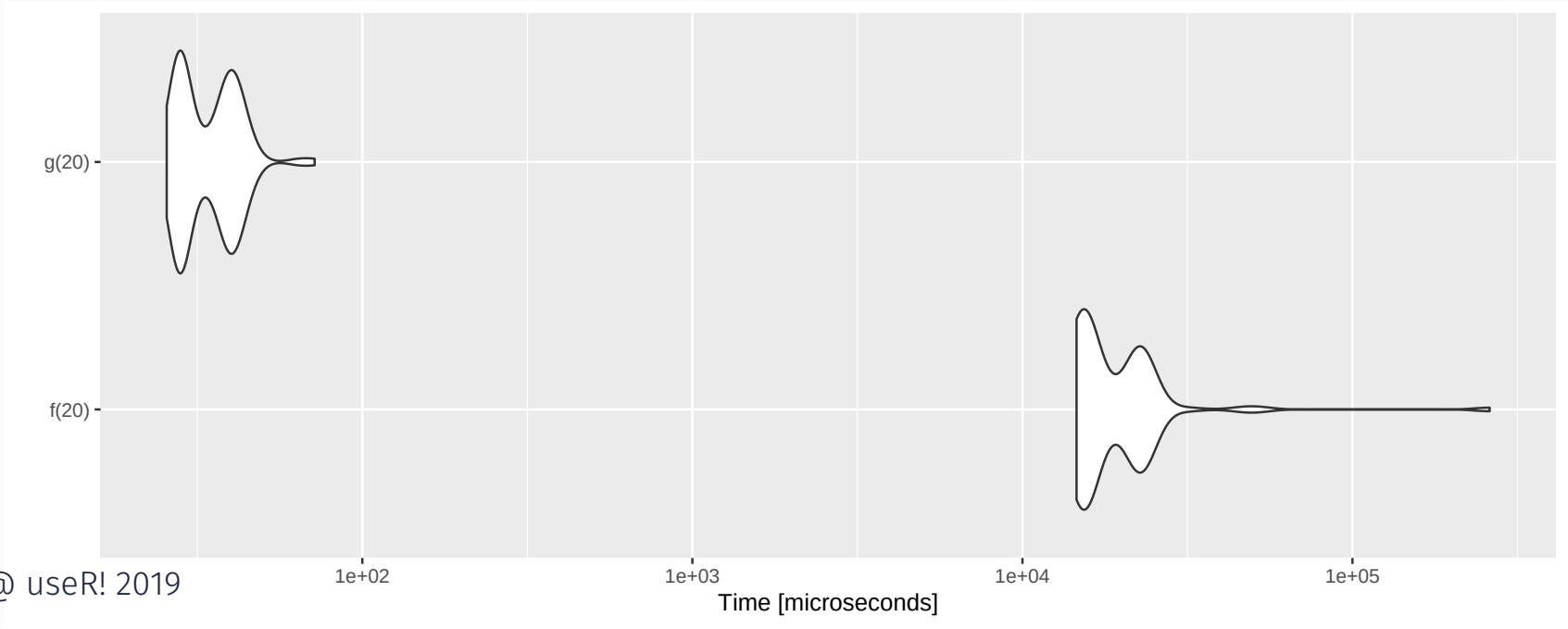
A nice gain of a few orders of magnitude.

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) 14588.41 15175.82 21357.65 15908.71 22670.03 260417.6   100   b
##  g(20)    25.55    28.13    34.95    33.94    40.06     71.7   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
```

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

C++ is a Better C

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;
};
```

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();
}
```

C++ and R Types

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.

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:

```
// [[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);
}
```

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
```

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
```

Sixth Exercise

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.

Two More Things on Rcpp Vectors

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.

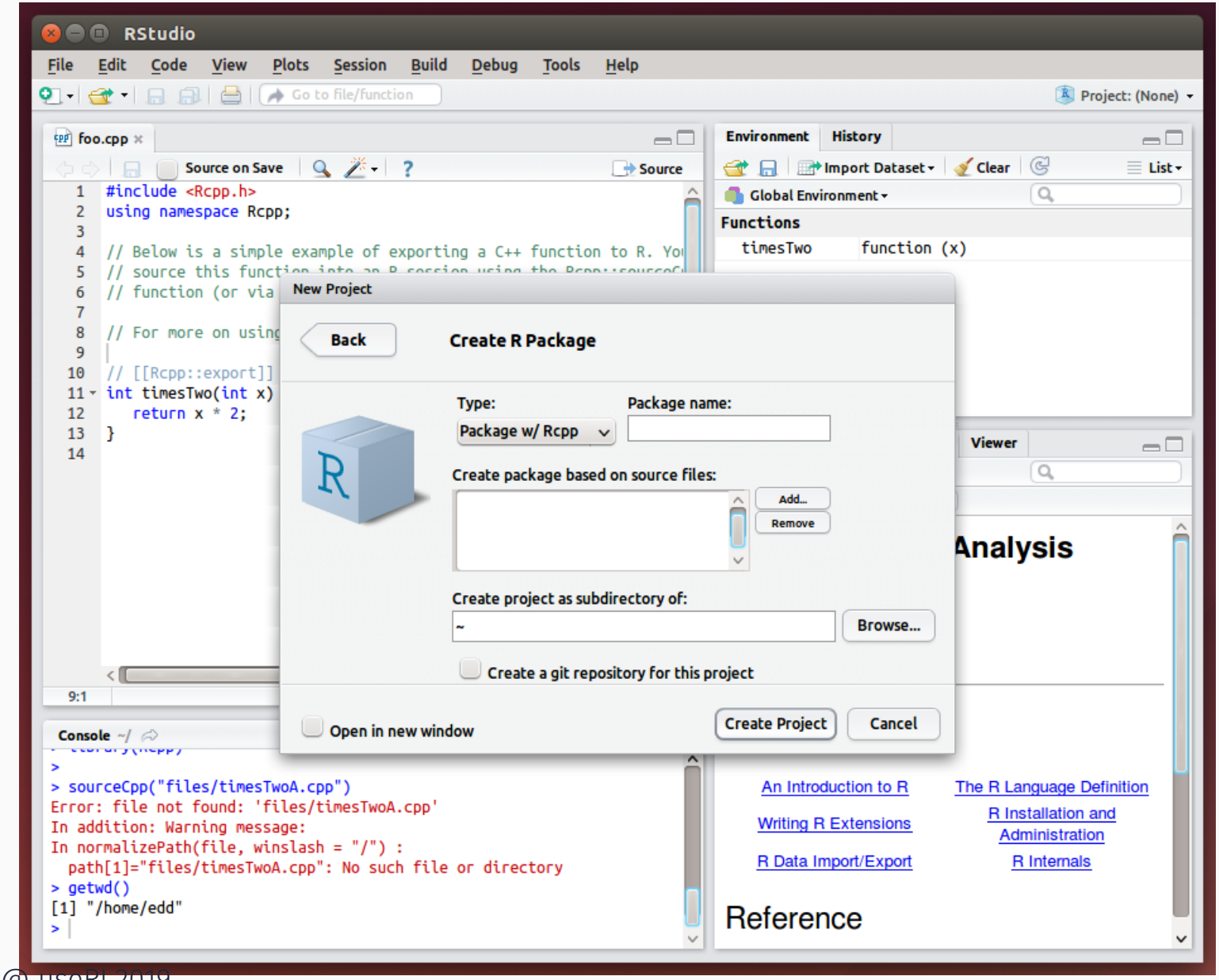
How: Packages

Basic Usage: 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 July 2019, there are 1692 CRAN and 176 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;
}
```

Packages and Rcpp

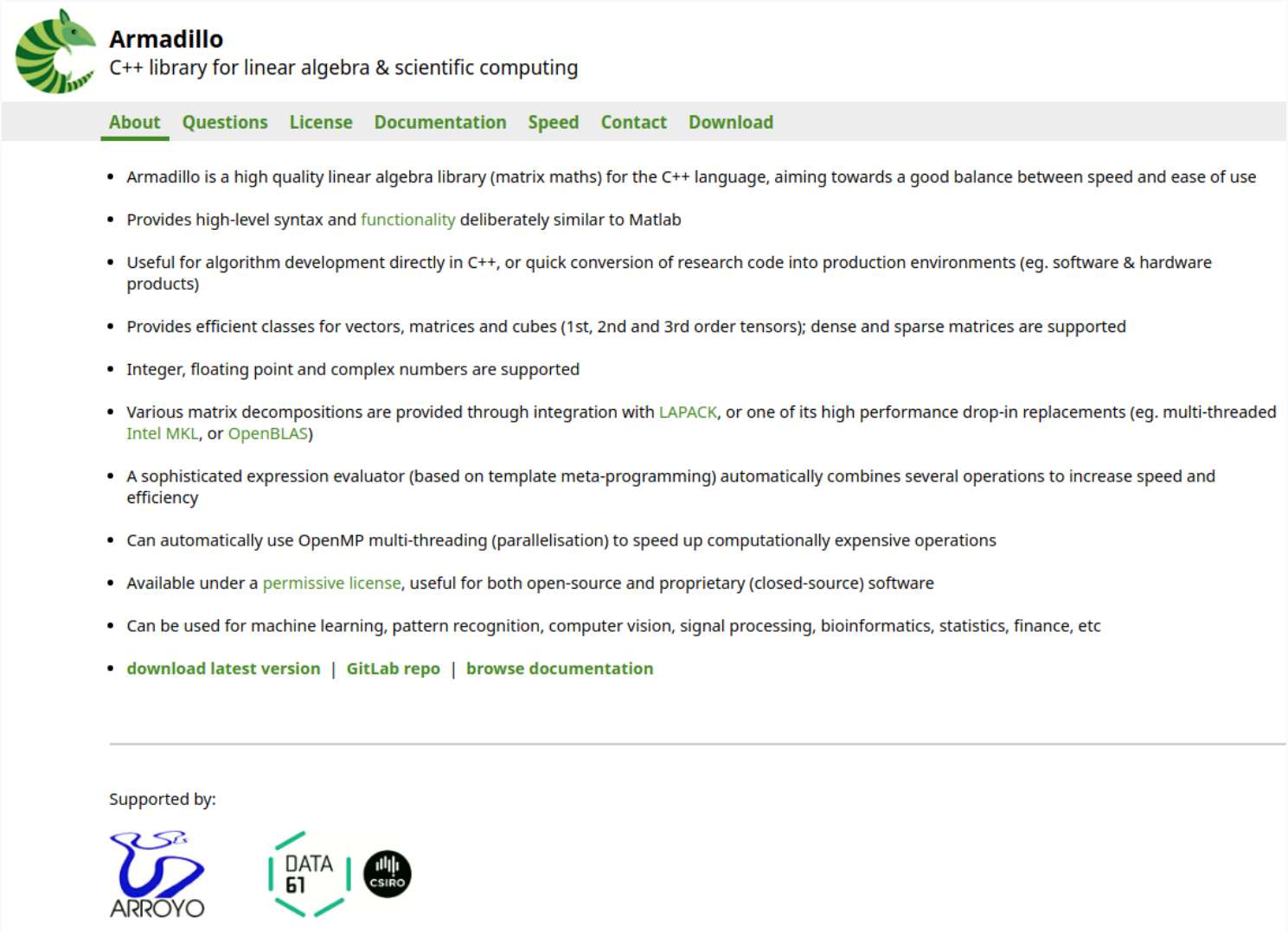
Two (or three) ways to link to external libraries

- *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



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:

ARROYO DATA 61 CSIRO

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`

Armadillo Highlights

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).

RcppArmadillo Highlights

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 – now over 600 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/arma_eigenvalues.cpp")
M <- cbind(c(1,-1), c(-1,1))
getEigenValues(M)
```

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

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

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

Seventh Exercise

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;
}
```

Package with RcppArmadillo

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
- One caveat on macOS is the need for `gfortran`, see online help

Third Example: FastLm

FastLm Case Study: Faster Linear Model with 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

```
#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
    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
}
```

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, 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
```

Benchmark

```
## continued from above
                     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:~$
```

Machine Learning

Overview

Among the almost 1700+ CRAN packages using Rcpp, many wrap wrap Machine Learning libraries algorithms.

Examples include:

- dlib based on (fairly large) DLib library
- RcppShark based on the Shark library
  (but archived in March 2018)
- RcppMLPACK based on an older MLPACK versions
  (newer version on GitHub only here)

We look at RcppMLPACK.

MLPACK

High-level:

- Written by Ryan Curtin et al, Georgia Tech
- Uses Armadillo, and like Armadillo, “feels right”
- Qiang Kou created ‘RcppMLPACK v1’, it is on CRAN

MLPACK ‘v2’

High-level:

- A few of us are trying to update RcppMLPACK to ‘v2’ (and ‘v3’)
- Instead of embedding, an external library is used
- This makes deployment a little tricker on Windows and macOS
- We are still waiting on macOS installation of libraries

List of Algorithms:

- Collaborative filtering (with many decomposition techniques)
- Decision stumps (one-level decision trees)
- Density estimation trees
- Euclidean minimum spanning tree calculation
- Gaussian mixture models
- Hidden Markov models
- Kernel Principal Components Analysis (optionally with sampling)
- k-Means clustering (with several accelerated algorithms)
- Least-angle regression (LARS/LASSO)
- Linear regression (simple least-squares)
- Local coordinate coding
- Locality-sensitive hashing for approximate nearest neighbor search
- Logistic regression
- Max-kernel search
- Naive Bayes classifier
- Nearest neighbor search with dual-tree algorithms
- Neighborhood components analysis
- Non-negative matrix factorization
- Perceptrons
- Principal components analysis (PCA)
- RADICAL (independent components analysis)
- Range search with dual-tree algorithms
- Rank-approximate nearest neighbor search
- Sparse coding with dictionary learning

RcppMLPACK: K-Means Example

```
#include "RcppMLPACK.h"

using namespace mlpack::kmeans;
using namespace Rcpp;

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

// [[Rcpp::export]]
List cppKmeans(const arma::mat& data, const int& clusters) {

    arma::Col<size_t> assignments;
    KMeans<> k;     // Initialize with the default arguments.
    k.Cluster(data, clusters, assignments);

    return List::create(Named("clusters") = clusters,
                        Named("result")   = assignments);
}
```

Timing

Table 1: Benchmarking result

test	replications	elapsed	relative	user.self	sys.self
mlKmeans(t(wine), 3)	100	0.028	1.000	0.028	0.000
kmeans(wine, 3)	100	0.947	33.821	0.484	0.424

Table taken 'as is' from RcppMLPACK vignette.

RcppMLPACK: Linear Regression Example

```
// [[Rcpp::depends(RcppMLPACK)]]
// [[Rcpp::plugins(openmp)]]
#include <RcppMLPACK.h>                  // MLPACK, Rcpp and RcppArmadillo

// particular algorithm used here
#include <mlpack/methods/linear_regression/linear_regression.hpp>

// [[Rcpp::export]]
arma::vec linearRegression(arma::mat& matX,
                           arma::vec& vecY,
                           const double lambda = 0.0,
                           const bool intercept = true) {

    matX = matX.t();
    mlpack::regression::LinearRegression lr(matX, vecY.t(), lambda, intercept);
    arma::rowvec fittedValues(vecY.n_elem);
    lr.Predict(matX,  fittedValues);
    return fittedValues.t();
}
```

RcppMLPACK: Linear Regression Example

```
suppressMessages(library(utils))
library(RcppMLPACK)
data("trees", package="datasets")
X <- with(trees, cbind(log(Girth), log(Height)))
y <- with(trees, log(Volume))
lmfit <- lm(y ~ X)
# summary(fitted(lmfit))

mlfit <- linearRegression(X, y)
# summary(mlfit)

all.equal(unname(fitted(lmfit)),  as.vector(mlfit))
```

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

RcppMLPACK: Logistic Regression Example

```
#include <RcppMLPACK.h>             // MLPACK, Rcpp and RcppArmadillo
#include <mlpack/methods/logistic_regression/logistic_regression.hpp>   // algo use here

// [[Rcpp::export]]
Rcpp::List logisticRegression(const arma::mat& train, const arma::irowvec& labels,
                              const Rcpp::Nullable<Rcpp::NumericMatrix>& test = R_NilValue) {

    // MLPACK wants Row<size_t> which is an unsigned representation that R does not have
    arma::Row<size_t> labelsur, resultsur;
    // TODO: check that all values are non-negative
    labelsur = arma::conv_to<arma::Row<size_t>>::from(labels);
    // Initialize with the default arguments. TODO: support more arguments>
    mlpack::regression::LogisticRegression<> lrc(train, labelsur);
    arma::rowvec parameters = lrc.Parameters();

    Rcpp::List return_val;
    if (test.isNotNull()) {
        arma::mat test2 = Rcpp::as<arma::mat>(test);
        lrc.Classify(test2, resultsur);
        arma::vec results = arma::conv_to<arma::vec>::from(resultsur);
        return_val = Rcpp::List::create(Rcpp::Named("parameters") = parameters,
                                        Rcpp::Named("results")    = results);
    } else {
        return_val = Rcpp::List::create(Rcpp::Named("parameters") = parameters);
    }
    return return_val;
}
```

RcppMLPACK: Linear Regression Example

```
suppressMessages(library(utils));
library(RcppMLPACK);
## part of  example(logisticRegression), test data not show for space constraint
data(trainSet)
mat <- t(trainSet[, -5])     ## train data, transpose and removing class labels
lab <- trainSet[, 5]         ## class labels for train set
logisticRegression(mat, lab)
```

```
## train
##    3.0000   3.0000   3.0000   3.0000   3.0000   2.0000   2.0000   3.0000   3.0000   3.0000   3.0000   3.0000
##    3.0000   4.0000   4.0000   3.0000   6.0000   4.0000   4.0000   3.0000   4.0000   4.0000   3.0000   6.0000
##    3.0000   4.0000   4.0000   4.0000   4.0000   4.0000   4.0000   3.0000   4.0000   4.0000   4.0000   4.0000
##    3.0000   3.0000   3.0000   3.0000   3.0000   3.0000   1.0000   2.0000   2.0000   2.0000   2.0000   2.0000
## labels
##         0        0        0        0        0        0        0        0        0        0        0        0
```

```
## $parameters
## [1] -11.0819909  13.9022481   0.8034972  -9.3485217 -13.0869968
```

RcppMLPACK: Nearest Neighbors Example

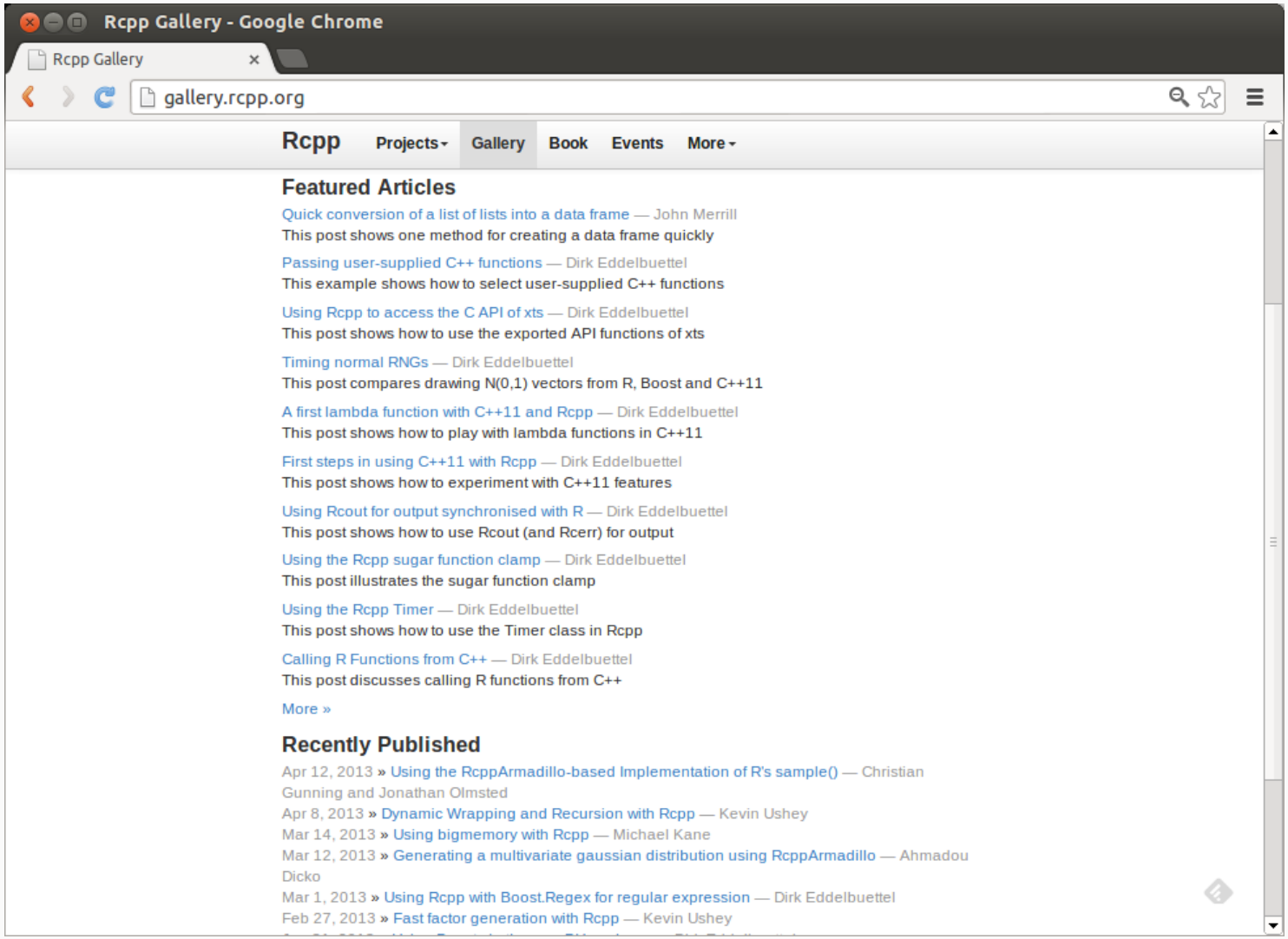
```
#include "RcppMLPACK.h"

using namespace Rcpp;
using namespace mlpack;           using namespace mlpack::neighbor;
using namespace mlpack::metric;   using namespace mlpack::tree;

// [[Rcpp::depends(RcppMLPACK)]]
// [[Rcpp::export]]
List nn(const arma::mat& data, const int k) {
    // using a test from MLPACK 1.0.10 file src/mlpack/tests/allknn_test.cpp
    CoverTree<LMetric<2>, FirstPointIsRoot,
              NeighborSearchStat<NearestNeighborSort> > tree =
        CoverTree<LMetric<2>, FirstPointIsRoot,
                  NeighborSearchStat<NearestNeighborSort> >(data);

    NeighborSearch<NearestNeighborSort, LMetric<2>,
                   CoverTree<LMetric<2>, FirstPointIsRoot,
                             NeighborSearchStat<NearestNeighborSort> > >
        coverTreeSearch(&tree, data, true);

    arma::Mat<size_t> coverTreeNeighbors;
    arma::mat coverTreeDistances;
    coverTreeSearch.Search(k, coverTreeNeighbors, coverTreeDistances);

    return List::create(Named("clusters") = coverTreeNeighbors,
                        Named("result")   = coverTreeDistances);
}
```

More

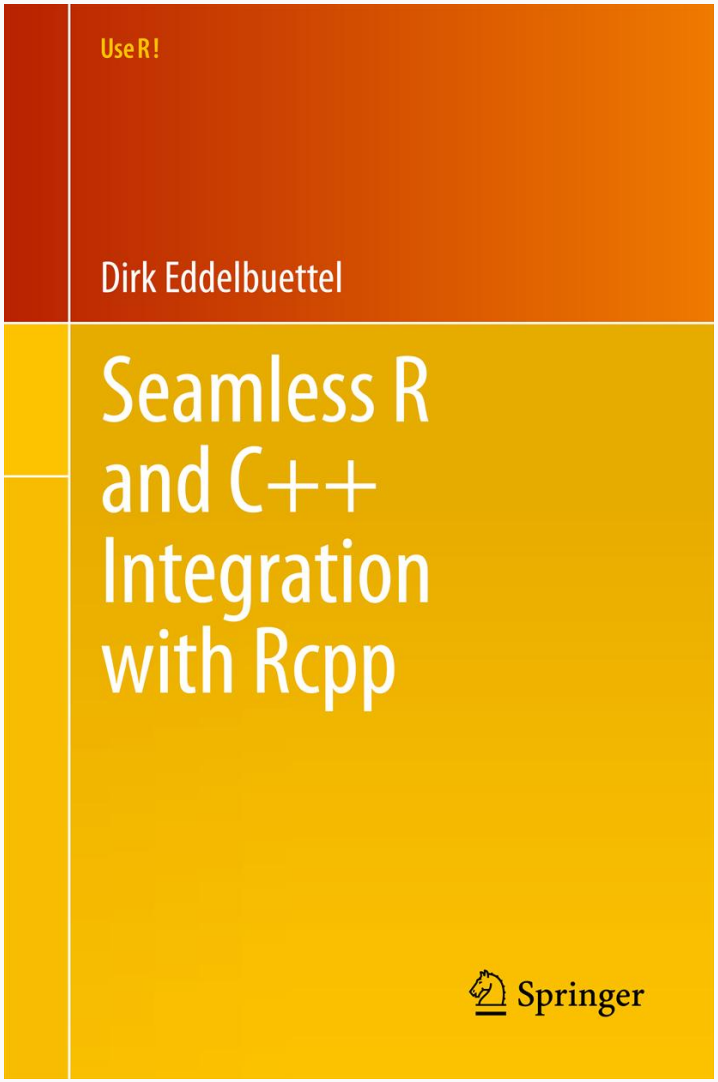
Where to go for next steps

- 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.

Rcpp Gallery



The Rcpp book



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

Appendix: More Rcpp Examples

Cumulative Sum: vector-cumulative-sum

A basic looped version:

```
#include <Rcpp.h>
#include <numeric>      // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
    double acc = 0;     // init an accumulator variable

    NumericVector res(x.size());  // init result vector

    for(int i = 0; i < x.size(); i++){
        acc += x[i];
        res[i] = acc;
    }
    return res;
}
```

Cumulative Sum: vector-cumulative-sum

An STL variant:

```
// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
    // initialize the result vector
    NumericVector res(x.size());
    std::partial_sum(x.begin(), x.end(), res.begin());
    return res;
}
```

Cumulative Sum: vector-cumulative-sum

Or just Rcpp sugar:

```
// [[Rcpp::export]]
NumericVector cumsum_sug(NumericVector x){
    return cumsum(x);  // compute + return result vector
}
```

Of course, all results are the same.

R Function Call 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)
*/
```

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

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

// [[Rcpp::export]]
NumericVector positives(NumericVector x) {
    return x[x > 0];
}

// [[Rcpp::export]]
List first_three(List x) {
    IntegerVector idx = IntegerVector::create(0, 1, 2);
    return x[idx];
}

// [[Rcpp::export]]
List with_names(List x, CharacterVector y) {
    return x[y];
}
```

Armadillo Eigenvalues: armadillo-eigenvalues

```
#include <RcppArmadillo.h>

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

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

Armadillo Eigenvalues: armadillo-eigenvalues

```
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 gets the same results (in reverse)
# and also returns the eigenvectors.
```

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

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

NumericVector createXts(int sv, int ev) {
    IntegerVector ind = seq(sv, ev);     // values

    NumericVector dv(ind);               // date(time)s == reals
    dv = dv * 86400;                     // scaled to days
    dv.attr("tzone")    = "UTC";         // index has attributes
    dv.attr("tclass")   = "Date";

    NumericVector xv(ind);               // data has same index
    xv.attr("dim")         = IntegerVector::create(ev-sv+1,1);
    xv.attr("index")       = dv;
    CharacterVector cls = CharacterVector::create("xts","zoo");
    xv.attr("class")       = cls;
    xv.attr(".indexCLASS") = "Date";
    // ... some more attributes ...
    return xv;
}
```

RcppParallel 1/3: parallel-matrix-transform

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

#include <cmath>
#include <algorithm>

// [[Rcpp::export]]
NumericMatrix matrixSqrt(NumericMatrix orig) {

  // allocate the matrix we will return
  NumericMatrix mat(orig.nrow(), orig.ncol());

  // transform it
  std::transform(orig.begin(), orig.end(), mat.begin(), ::sqrt);

  // return the new matrix
  return mat;
}
```

RcppParallel 2/3: parallel-matrix-transform

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

struct SquareRoot : public Worker {

   const RMatrix<double> input;   // source matrix
   RMatrix<double> output;        // destination matrix

   // initialize with source and destination
   SquareRoot(const NumericMatrix input, NumericMatrix output)
      : input(input), output(output) {}

   // take the square root of the range of elements requested
   void operator()(std::size_t begin, std::size_t end) {
      std::transform(input.begin() + begin,  input.begin() + end,
                     output.begin() + begin,  ::sqrt);
   }
};
```

RcppParallel 3/3: parallel-matrix-transform

```
// [[Rcpp::export]]
NumericMatrix parallelMatrixSqrt(NumericMatrix x) {

  // allocate the output matrix
  NumericMatrix output(x.nrow(), x.ncol());

  // SquareRoot functor (pass input and output matrixes)
  SquareRoot squareRoot(x, output);

  // call parallelFor to do the work
  parallelFor(0, x.length(), squareRoot);

  // return the output matrix
  return output;
}
```