Introduction to Rcpp:

From Simple Examples to Machine Learning

Pre-Conference Tutorial

Dirk Eddelbuettel

useR! 2017

July 4, 2017

Ketchum Trading; Debian and R Projects

Overview

High-level motivation: Three main questions

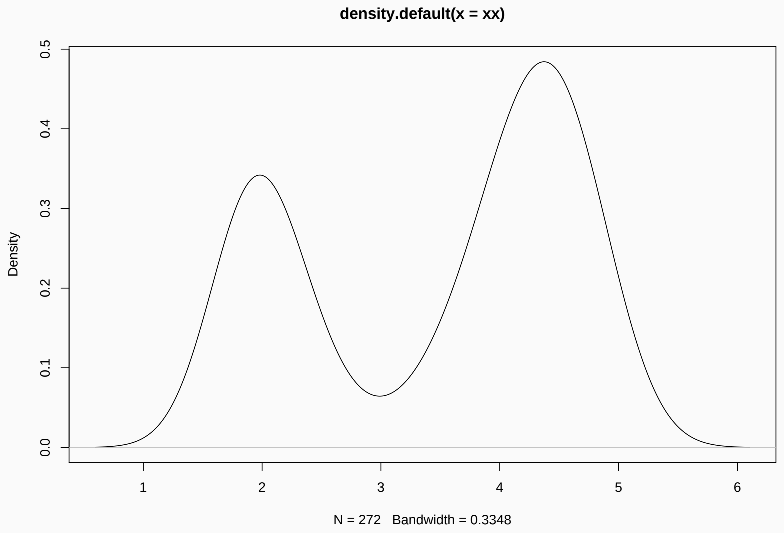
- Why ? *Several reasons discussed next*
- How ? *Rcpp details, usage, tips, ...*
- What ? *We will cover examples.*

Why R?

A Simple Example

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

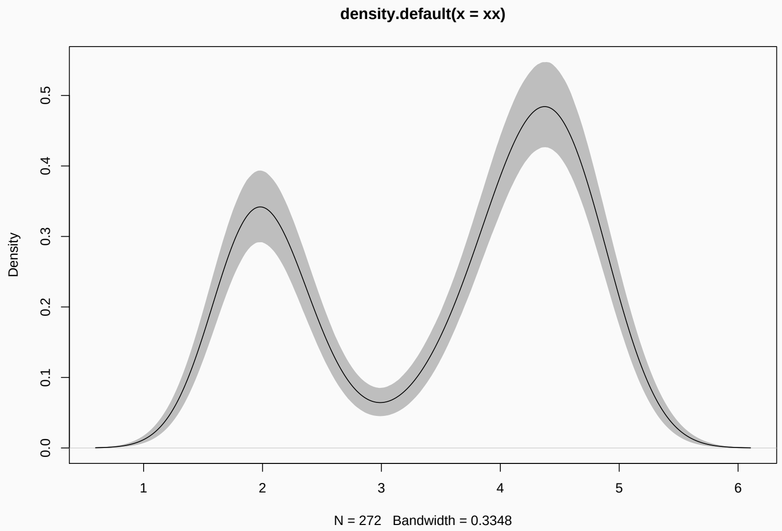
A Simple Example



A Simple Example - Refined

```
xx <- faithful[,"eruptions"]
fit1 <- density(xx)
fit2 <- replicate(10000, {
    x <- sample(xx,replace=TRUE);
    density(x, from=min(fit1$x), to=max(fit1$x))$y
})
fit3 <- apply(fit2, 1, quantile,c(0.025,0.975))
plot(fit1, ylim=range(fit3))
polygon(c(fit1$x,rev(fit1$x)), c(fit3[1,],rev(fit3[2,])),
    col='grey', border=F)
lines(fit1)
```

A Simple Example - Refined



So Why R?

R enables us to

- work interactively
- explore and visualize data
- access, retrieve and/or generate data
- summarize and report into pdf, html, ...

making it the key language for statistical computing, and a preferred environment for many data analysts.

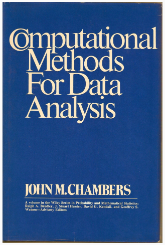
So Why R?

R has always been extensible via

- C via a bare-bones interface described in *Writing R Extensions*
- Fortran which is also used internally by R
- Java via `rJava` by Simon Urbanek
- C++ but essentially at the bare-bones level of C

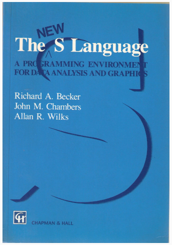
So while *in theory* this always worked – it was tedious *in practice*

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


Computational Methods For Data Analysis

JOHN M. CHAMBERS

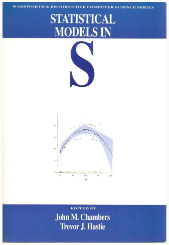
A volume in the Wiley Series in Probability and Mathematical Statistics: Ralph A. Bradley, J. Stuart Hunter, David G. Kendall, and Geoffrey S. Watson—Advisory Editors


The NEW S Language

A PROGRAMMING ENVIRONMENT FOR DATA ANALYSIS AND GRAPHICS

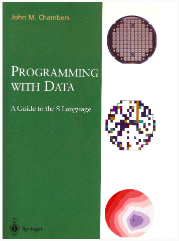
Richard A. Becker
John M. Chambers
Allan R. Wilks

CHAPMAN & HALL


WADSWORTH & BROOKS/COLE COMPUTER SCIENCE SERIES

STATISTICAL MODELS IN S

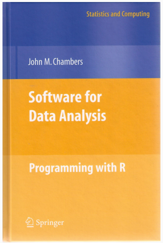
EDITED BY
John M. Chambers
Trevor J. Hastie


John M. Chambers

PROGRAMMING WITH DATA

A Guide to the S Language

Springer

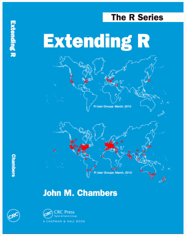

Statistics and Computing

John M. Chambers

Software for Data Analysis

Programming with R

Springer


The R Series

Extending R

John M. Chambers

CRC Press

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

Why Extend R?

Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

> *Since the core of R is in fact a program written in the C language, it's not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*

Why Extend R?

Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

> *Since the core of R is in fact a program written in the C language, it's not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*

Why Extend R?

Chambers proceeds with this rough map of the road ahead:

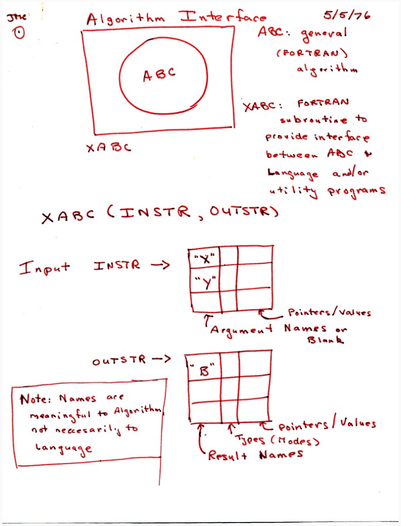
- Against:
    - It's more work
    - Bugs will bite
    - Potential platform dependency
    - Less readable software
- In Favor:
    - New and trusted computations
    - Speed
    - Object references

Why Extend R?

The *Why?* boils down to:

- speed: Often a good enough reason for us … and a focus for us in this workshop.
- new things: We can bind to libraries and tools that would otherwise be unavailable in R
- references: Chambers quote from 2008 foreshadowed the work on *Reference Classes* now in R and built upon via Rcpp Modules, Rcpp Classes (and also RcppR6)

Bell Labs, May 1976



JMC ①

Algorithm Interface 5/5/76

ABC

XABC

ABC: general (FORTRAN) algorithm

XABC: FORTRAN subroutine to provide interface between ABC & Language and/or utility programs

XABC (INSTR, OUTSTR)

Input INSTR →

"X"

"Y"

Argument Names or Blank

Pointers/Values

OUTSTR →

"B"

Result Names

Types (Modes)

Pointers/Values

Note: Names are meaningful to Algorithm, not neccesarily to Language

Interface Vision

R offers us the best of both worlds:

- Compiled code with
  - Access to proven libraries and algorithms in C/C++/Fortran
  - Extremely high performance (in both serial and parallel modes)
- Interpreted code with
  - A high-level language made for *Programming with Data*
  - An interactive workflow for data analysis
  - Support for rapid prototyping, research, and experimentation

And Why C++?

- Asking Google leads to tens of million of hits.
- Wikipedia: *C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language*
- C++ is industrial-strength, vendor-independent, widely-used, and *still evolving*
- In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API
- As a widely used language it also has good tool support (debuggers, profilers, code analysis)

Why C++?

Scott Meyers: *View C++ as a federation of languages*

- *C* provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.
- *Object-Oriented C++* (maybe just to provide endless discussions about exactly what OO is or should be)
- *Templated C++* which is mighty powerful; template meta programming unequalled in other languages.
- *The Standard Template Library* (STL) is a specific template library which is powerful but has its own conventions.
- *C++11* and C++14 (and beyond) add enough to be called a fifth language.

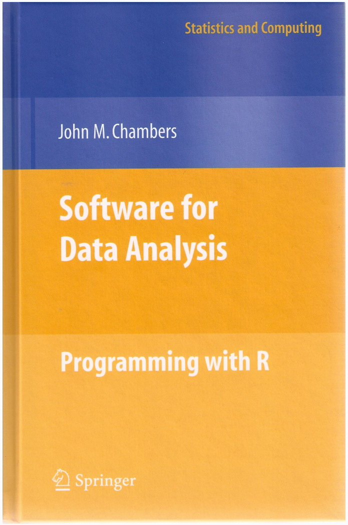
NB: Meyers original list of four languages appeared years before C++11.

Why C++?

- Mature yet current
- Strong performance focus:
    - *You don't pay for what you don't use*
    - *Leave no room for another language between the machine level and C++*
- Yet also powerfully abstract and high-level
- C++11 + C++14 are a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded

Interlude

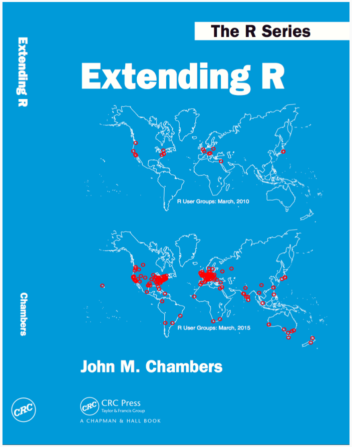
John Chambers



Chambers (2008) Software For Data Analysis

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

John Chambers



Chambers (2016) Extending R

An entire book about this with *concrete* Python, Julia and C++ code and examples

John Chambers

Chambers 2016, Chapter 1

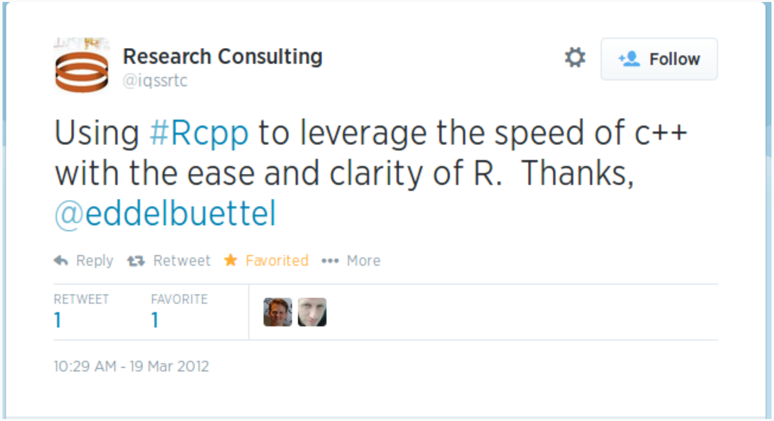
- *Everything that exists in R is an object*
- *Everything happens in R is a function call*
- *Interfaces to other software are part of R*

John Chambers

Chambers 2016, 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.*

Why Rcpp? Some Tweets

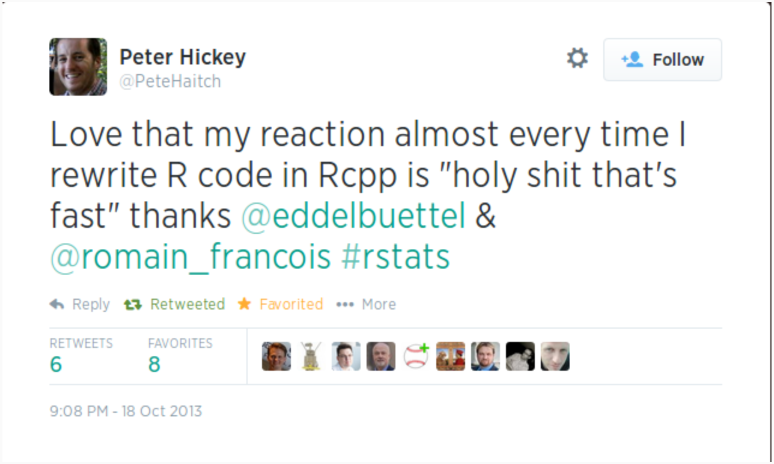


Research Consulting
@iqssrtc

Using #Rcpp to leverage the speed of c++ with the ease and clarity of R. Thanks, @eddelbuettel

RETWEET 1 FAVORITE 1

10:29 AM - 19 Mar 2012



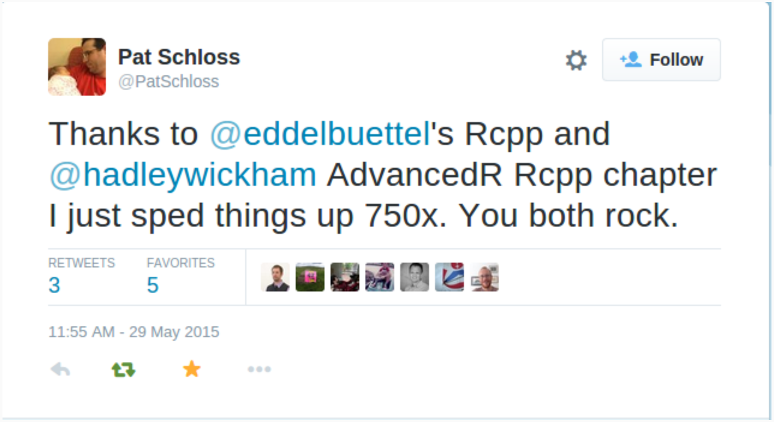
Peter Hickey
@PeteHaitch

Follow

Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois #rstats

RETWEETS 6
FAVORITES 8

9:08 PM - 18 Oct 2013



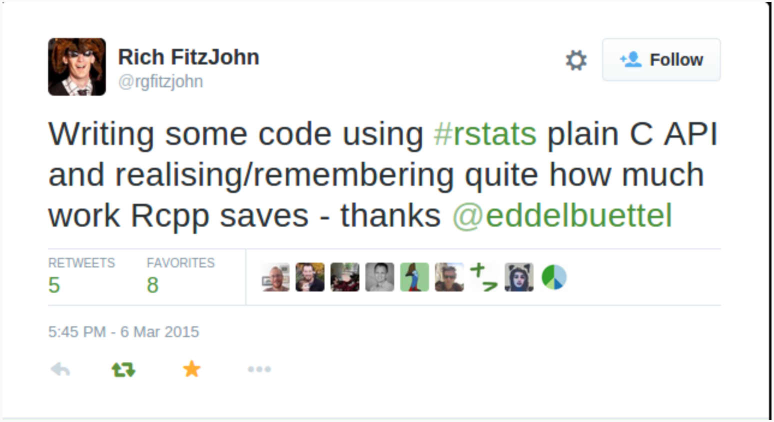
Pat Schloss
@PatSchloss

Follow

Thanks to @eddelbuettel's Rcpp and @hadleywickham AdvancedR Rcpp chapter I just sped things up 750x. You both rock.

RETWEETS 3
FAVORITES 5

11:55 AM - 29 May 2015



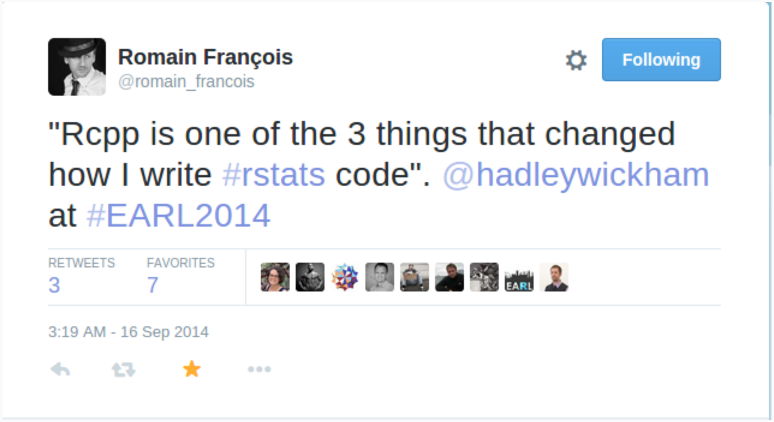
Rich FitzJohn
@rgfitzjohn

Follow

Writing some code using #rstats plain C API and realising/remembering quite how much work Rcpp saves - thanks @eddelbuettel

RETWEETS 5 FAVORITES 8

5:45 PM - 6 Mar 2015

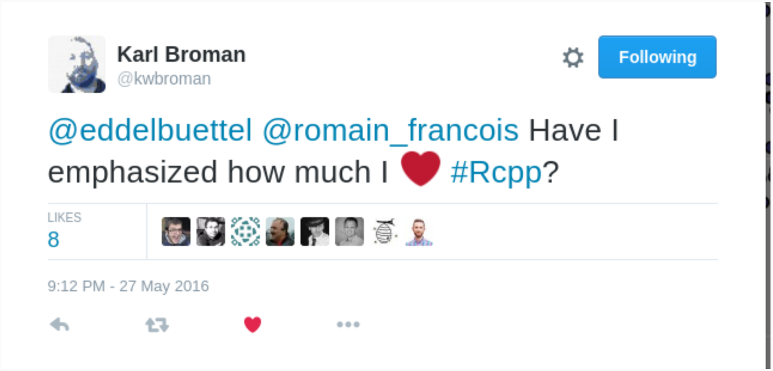


Romain François
@romain_francois

"Rcpp is one of the 3 things that changed how I write #rstats code". @hadleywickham at #EARL2014

RETWEETS 3
FAVORITES 7

3:19 AM - 16 Sep 2014

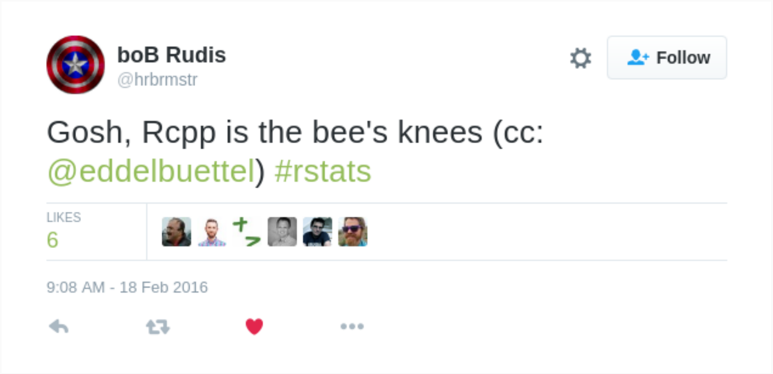


Karl Broman
@kwbroman

@eddelbuettel @romain_francois Have I emphasized how much I ❤ #Rcpp?

LIKES
8

9:12 PM - 27 May 2016

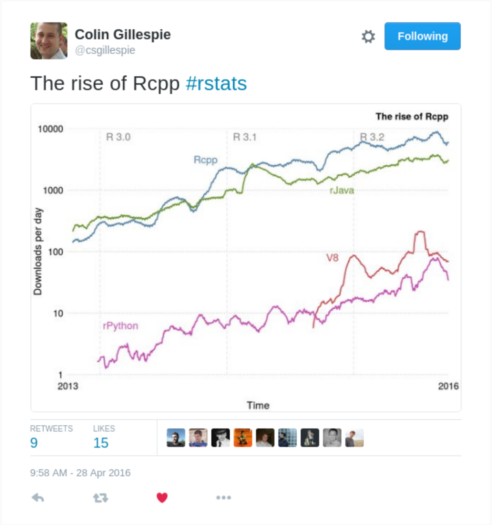


boB Rudis
@hrbrmstr

Gosh, Rcpp is the bee's knees (cc: @eddelbuettel) #rstats

LIKES
6

9:08 AM - 18 Feb 2016

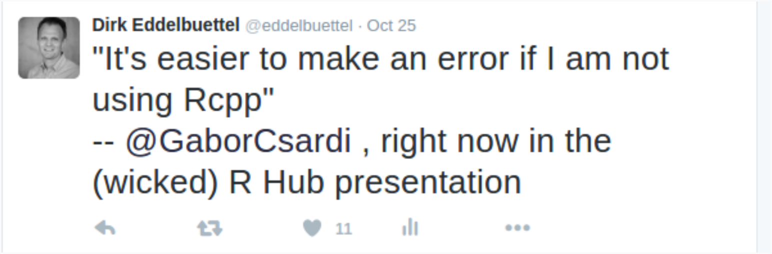


Colin Gillespie
@csgillespie

The rise of Rcpp #rstats

RETWEETS 9 LIKES 15

9:58 AM - 28 Apr 2016



Dirk Eddelbuettel @eddelbuettel · Oct 25

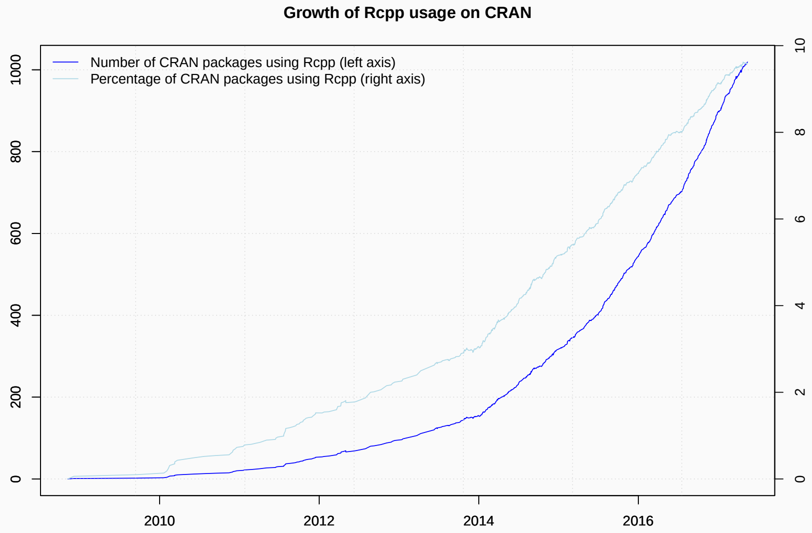
"It's easier to make an error if I am not using Rcpp"
-- @GaborCsardi , right now in the (wicked) R Hub presentation

Why Rcpp?

Key points

- *Easy to learn* as it really does not have to be that complicated – we will see numerous few examples
- *Easy to use* as it avoids build and OS system complexities thanks to the R infrastrucure
- *Expressive* as it allows for *vectorised* C++ using *Rcpp Sugar*
- *Seamless* access to all R objects: vector, matrix, list, S3/S4/RefClass, Environment, Function, …
- *Speed gains* for a variety of tasks Rcpp excels precisely where R struggles: loops, function calls, …
- *Extensions* greatly facilitates access to external libraries using eg *Rcpp modules*

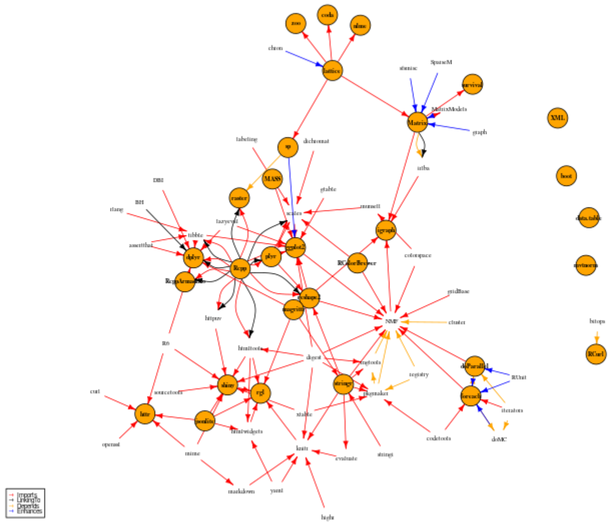
Who Uses Rcpp?



Data current as of June 17, 2017.

Pagerank

Top 30 packages by page rank



CRAN Proportion

```
db <- tools::CRAN_package_db()   # R 3.4.0 or later
dim(db)
```

```
## [1] 10926    65
```

```
## all Rcpp reverse depends
(c(n_rcpp <- length(tools::dependsOnPkgs("Rcpp", recursive=FALSE,
                                         installed=db)),
   n_compiled <- table(db[, "NeedsCompilation"])[["yes"]]))
```

```
## [1] 1069 2919
```

```
## Rcpp percentage of packages with compiled code
n_rcpp / n_compiled
```

```
## [1] 0.3662213
```

Speed

Speed Example 1 (due to Christian Robert)

Five different ways to compute $1/(1+x)$:

```
f <- function(n, x=1) for(i in 1:n) x <- 1/(1+x)
g <- function(n, x=1) for(i in 1:n) x <- (1/(1+x))
h <- function(n, x=1) for(i in 1:n) x <- (1+x)^(-1)
j <- function(n, x=1) for(i in 1:n) x <- {1/{1+x}}
k <- function(n, x=1) for(i in 1:n) x <- 1/{1+x}
library(rbenchmark)
N <- 1e5
benchmark(f(N,1),g(N,1),h(N,1),j(N,1),k(N,1),order="relative")[
```

Speed Example 1 (due to Christian Robert)

```
##       test replications elapsed relative
## 1 f(N, 1)          100   0.555    1.000
## 2 g(N, 1)          100   0.556    1.002
## 5 k(N, 1)          100   0.572    1.031
## 4 j(N, 1)          100   0.597    1.076
## 3 h(N, 1)          100   0.666    1.200
```

Speed Example 1 (due to Christian Robert)

Adding a C++ variant is easy:

```
cppFunction("
    double m(int n, double x) {
        for (int i=0; i<n; i++)
             x = 1 / (1+x);
        return x;
    }"
)
```

(We will learn more about `cppFunction()` later).

Speed Example 1 (due to Christian Robert)

```
##       test replications elapsed relative
## 6 m(N, 1)          100   0.074    1.000
## 2 g(N, 1)          100   0.541    7.311
## 1 f(N, 1)          100   0.554    7.486
## 5 k(N, 1)          100   0.582    7.865
## 4 j(N, 1)          100   0.629    8.500
## 3 h(N, 1)          100   0.761   10.284
```

Speed Example 2 (due to StackOverflow)

Consider a function defined as

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

Speed Example 2 (due to StackOverflow)

R implementation and use:

```r
f <- function(n) {
    if (n < 2) return(n)
    return(f(n-1) + f(n-2))
}

## Using it on first 11 arguments
sapply(0:10, f)
```

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

Speed Example 2 (due to StackOverflow)

Timing:

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

```
##     test replications elapsed relative
## 1 f(10)          100   0.011    1.000
## 2 f(15)          100   0.115   10.455
## 3 f(20)          100   1.388  126.182
```

Speed Example 2 (due to StackOverflow)

```
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)
```

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

Speed Example 2 (due to StackOverflow)

Timing:

```
Rcpp::cppFunction("int g(int n) {
   if (n < 2) return(n);
   return(g(n-1) + g(n-2)); }")
library(rbenchmark)
benchmark(f(20), g(20), order="relative")[,1:4]
```

```
##    test replications elapsed relative
## 2 g(20)          100   0.008     1.00
## 1 f(20)          100   1.258   157.25
```

A nice gain of a few orders of magnitude.

Another Angle on Speed

Run-time performance is just one example.

Time to code is another metric.

We feel quite strongly that helps you code more succinctly, leading to fewer bugs and faster development.

A good environment helps. RStudio integrates R and C++ development quite nicely (eg the compiler error message parsing is very helpful) and also helps with package building.

What Next ?

Programming with C++

- C++ Basics
- Debugging
- Best Practices

and then on to Rcpp itself

Compiled not Interpreted

Need to compile and link

```
#include <cstdio>

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

Compiled not Interpreted

Or streams output rather than `printf`

```
#include <iostream>

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

Compiled not Interpreted

`g++ -o` will compile and link

We will now look at an examples with explicit linking.

Compiled not Interpreted

```
#include <cstdio>

#define MATHLIB_STANDALONE
#include <Rmath.h>

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

Compiled not Interpreted

We may need to supply:

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

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

Statically Typed

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

C++ is a Better C

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

Object-Oriented

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

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

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

Object-Oriented

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

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

Generic Programming and the STL

The STL promotes *generic* programming.

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

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

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

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

Generic Programming and the STL

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

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

Generic Programming and the STL

Another key STL part are *algorithms*:

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

Some other STL algorithms are

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

Template Programming

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

One of the simplest template examples is

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

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

Template Programming

Another template example is a class squaring its argument:

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

which can be used along with STL algorithms:

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

Further Reading

Books by Meyers are excellent

I also like the (free) C++ Annotations

C++ FAQ

Resources on StackOverflow such as

- general info and its links, eg
- booklist

Debugging

Some tips:

- Generally painful, old-school `printf()` still pervasive
- Debuggers go along with compilers: `gdb` for `gcc` and `g++`; `lldb` for the clang / llvm family
- Extra tools such as `valgrind` helpful for memory debugging
- “Sanitizer” (ASAN/UBSAN) in newer versions of `g++` and `clang++`

Best Practices

Version control: `git` or `svn` highly recommended

Editor: *in the long-run*, recommended to learn productivity tricks for one editor: emacs, vi, eclipse, RStudio, ...

Rcpp Usage

Basic Usage: evalCpp()

`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
```

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;
        return x;
}", plugins=c("cpp11"))
exampleCpp11()  # same identifier as C++ function
```

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).

Rcpp Gallery

Overview

- The *Rcpp Gallery* at http://gallery.rcpp.org provides over one-hundred ready-to-run and documented examples.
- It is built on a blog-alike backend in a repository hosted at GitHub.
- You can clone the repository, or just download examples one-by-one.

Simple 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 var
    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:

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

// [[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/

Sugar:

```
// [[Rcpp::export]]
NumericVector cumsum3(NumericVector x) {
    return cumsum(x);  // compute + return result
}
```

Calling an R Function: r-function-from-c++/

```cpp
#include <Rcpp.h>

using namespace Rcpp;

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

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];
}
```

Vector Subsetting: armadillo-subsetting/

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

// [[Rcpp::export]]
arma::mat matrixSubset(arma::mat M) {
    // logical conditionL where is transpose larger?
    arma::umat a = trans(M) > M;
    arma::mat  N = arma::conv_to<arma::mat>::from(a);
    return N;
}

// [[Rcpp::export]]
arma::vec matrixSubset2(arma::mat M) {
    arma::mat Z = M * M.t();
    arma::vec v = Z.elem( arma::find( Z >= 100 ) );
    return v;
}
```

Boost

Boost via BH: a-first-boost-example/

```
// [[Rcpp::depends(BH)]]
#include <Rcpp.h>
#include <boost/math/common_factor.hpp>

// [[Rcpp::export]]
int computeGCD(int a, int b) {
    return boost::math::gcd(a, b);
}

// [[Rcpp::export]]
int computeLCM(int a, int b) {
    return boost::math::lcm(a, b);
}
```

Boost via BH: a-second-boost-example/

```
// [[Rcpp::depends(BH)]]
#include <Rcpp.h>
#include <boost/lexical_cast.hpp>
using boost::lexical_cast;
using boost::bad_lexical_cast;

// [[Rcpp::export]]
std::vector<double> lexicalCast(std::vector<std::string> v) {
    std::vector<double> res(v.size());
    for (int i=0; i<v.size(); i++) {
        try {
            res[i] = lexical_cast<double>(v[i]);
        } catch(bad_lexical_cast &) {
            res[i] = NA_REAL;
        }
    }
    return res;
}
// R> lexicalCast(c("1.23", ".4", "1000", "foo", "42", "pi/4")(
// [1]    1.23    0.40 1000.00      NA   42.00      NA
```

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::Date(ymd.year, ymd.month, ymd.day);
}
```

Boost via BH: boost-foreach/

```cpp
#include <Rcpp.h>
#include <boost/foreach.hpp>
using namespace Rcpp;
// [[Rcpp::depends(BH)]]

// the C-style upper-case macro name is a bit ugly
#define foreach BOOST_FOREACH

// [[Rcpp::export]]
NumericVector square( NumericVector x ) {

    // elem is a reference to each element in x
    // we can re-assign to these elements as well
    foreach( double& elem, x ) {
        elem = elem*elem;
    }
    return x;
}
```

Boost via BH: boost-regular-expressions/

NB: Use `Sys.setenv("PKG_LIBS"="-lboost_regex")`

```
// boost.org/doc/libs/1_53_0/libs/regex/example/snippets/credit_card_example.cpp
#include <Rcpp.h>
#include <string>
#include <boost/regex.hpp>

bool validate_card_format(const std::string& s) {
    static const boost::regex e("(\\d{4}[- ]){3}\\d{4}");
    return boost::regex_match(s, e);
}

// [[Rcpp::export]]
std::vector<bool> regexDemo(std::vector<std::string> s) {
    int n = s.size();
    std::vector<bool> v(n);
    for (int i=0; i<n; i++)
        v[i]  = validate_card_format(s[i]);
    return v;
}
```

XPtr

Passing an XPtr: `passing-cpp-function-pointers/`

Consider two functions modifying a given Armadillo vector:

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

using namespace arma;
using namespace Rcpp;

vec fun1_cpp(const vec& x) {    // a first function
    vec y = x + x;
    return (y);
}

vec fun2_cpp(const vec& x) {    // and a second function
    vec y = 10*x;
    return (y);
}
```

Passing an XPtr: `passing-cpp-function-pointers/`

Using `typedef` to declare `funcPtr` as an interface to a function taking and returning a vector — and defining a function returning a function pointer given a string argument

```
typedef vec (*funcPtr)(const vec& x);

// [[Rcpp::export]]
XPtr<funcPtr> putFunPtrInXPtr(std::string fstr) {
    if (fstr == "fun1")
        return(XPtr<funcPtr>(new funcPtr(&fun1_cpp)));
    else if (fstr == "fun2")
        return(XPtr<funcPtr>(new funcPtr(&fun2_cpp)));
    else
        // runtime err.: NULL no XPtr
        return XPtr<funcPtr>(R_NilValue);
}
```

Passing an XPtr: `passing-cpp-function-pointers/`

Next we create a function calling the supplied function on a given vector by ‘unpacking’ the function pointer:

```
// [[Rcpp::export]]
vec callViaXPtr(const vec x, SEXP xpsexp) {
    XPtr<funcPtr> xpfun(xpsexp);
    funcPtr fun = *xpfun;
    vec y = fun(x);
    return (y);
}
```

Passing an XPtr: `passing-cpp-function-pointers/`

Putting it together:

```r
# this gets us a function
fun <- putFunPtrInXPtr(”fun1”)
# and pass it down to C++ to
# have it applied on given vector
callViaXPtr(1:4, fun)
```

This mechanism is generic and can be used for objective functions, gradients, samplers, … to operate at C++ speed on user-supplied C++ functions.

Plugins

Plugin support in Rcpp

```r
# setup plugins environment
.plugins <- new.env()
# built-in C++11 plugin
.plugins[["cpp11"]] <- function() {
    if (getRversion() >= "3.1")
        list(env = list(USE_CXX1X = "yes"))
    else if (.Platform$OS.type == "windows")
        list(env = list(PKG_CXXFLAGS = "-std=c++0x"))
    else
        list(env = list(PKG_CXXFLAGS ="-std=c++11"))
}
# built-in OpenMP++11 plugin
.plugins[["openmp"]] <- function() {
    list(env = list(PKG_CXXFLAGS="-fopenmp", PKG_LIBS="-fopenmp"))
}

# register a plugin
registerPlugin <- function(name, plugin) {
    .plugins[[name]] <- plugin
}
```

C++11 AUTO: `first-steps-with-C++11/`

```
#include <Rcpp.h>

// Enable C++11 via this plugin
// [[Rcpp::plugins(”cpp11”)]]

// [[Rcpp::export]]
int useAuto() {
    auto val = 42;       // val will be of type int
    return val;
}
```

C++11 INITLIST: first-steps-with-C++11/

```cpp
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

// [[Rcpp::export]]
std::vector<std::string> useInitLists() {
    std::vector<std::string> vec =
        {"larry", "curly", "moe"};
    return vec;
}
```

C++11 RANGE: first-steps-with-C++11/

```
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

// [[Rcpp::export]]
int simpleProd(std::vector<int> vec) {
    int prod = 1;
    for (int &x : vec) {       // loop over all values of vec
       prod *= x;              // access each elem., comp. proc
    }
    return prod;
}
```

C++11 LAMBDA: simple-lambda-func-c++11/

```
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

// [[Rcpp::export]]
std::vector<double>
transformEx(const std::vector<double>& x) {
    std::vector<double> y(x.size());
    std::transform(x.begin(), x.end(), y.begin(),
                   [](double x) { return x*x; } );
    return y;
}
```

OpenMP: using-rcppprogress/

We start we with (somewhat boring/made-up) slow double-loop:

```
#include <Rcpp.h>


// [[Rcpp::export]]
double long_computation(int nb) {
    double sum = 0;
    for (int i = 0; i < nb; ++i) {
        for (int j = 0; j < nb; ++j) {
            sum += R::dlnorm(i+j, 0.0, 1.0, 0);
        }
    }
    return sum + nb;
}
```

OpenMP: using-rcppprogress/

```
// [[Rcpp::plugins("openmp")]]
#include <Rcpp.h>

// [[Rcpp::export]]
double long_computation_omp(int nb, int threads=1) {
#ifdef _OPENMP
    if (threads > 0) omp_set_num_threads( threads );
    REprintf("Number of threads=%i\n", omp_get_max_threads());
#endif

    double sum = 0;
#pragma omp parallel for schedule(dynamic)
    for (int i = 0; i < nb; ++i) {
        double thread_sum = 0;
        for (int j = 0; j < nb; ++j) {
            thread_sum += R::dlnorm(i+j, 0.0, 1.0, 0);
        }
        sum += thread_sum;
    }
    return sum + nb;
}
```

OpenMP: using-rcppprogress/

Even on my laptop gains can be seen:

```
R> sourceCpp("code/openmpEx.cpp")
R> system.time(long_computation(1e4))
   user  system elapsed
 22.436   0.000  22.432
R> system.time(long_computation_omp(1e4,4))
Number of threads=4
   user  system elapsed
 25.432   0.076   7.046
R>
```

RcppParallel

Parallel Matrix Transform: parallel-matrix-transform/

```cpp
#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;
}
```

Parallel Matrix Transform: `parallel-matrix-transform/`

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

PARALLEL MATRIX TRANSFORM: `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;
}
```

Applications

Key Rcpp Application Packages

Overview

As of late June 2017, around 1069 packages on CRAN are using Rcpp

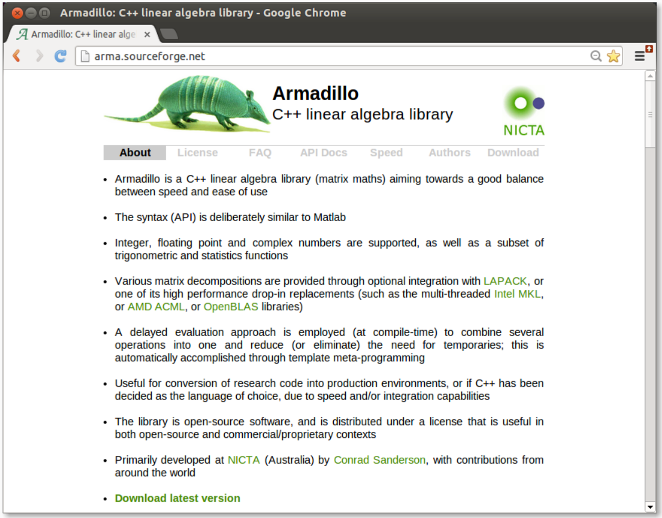
Single biggest “application” is the RcppArmadillo package for linear algebra with around 350

RcppEigen another important package used by around 100 packages including `lme4` and `RStan`

RcppGSL offers vector and matrix classes for the GSL, a popular scientific library

RcppArmadillo

Armadillo


Armadillo: C++ linear algebra library - Google Chrome
Armadillo: C++ linear alge
arma.sourceforge.net
NICTA
About
License
FAQ
API Docs
Speed
Authors
Download

Armadillo

C++ linear algebra library

- Armadillo is a C++ linear algebra library (matrix maths) aiming towards a good balance between speed and ease of use
- The syntax (API) is deliberately similar to Matlab
- Integer, floating point and complex numbers are supported, as well as a subset of trigonometric and statistics functions
- Various matrix decompositions are provided through optional integration with LAPACK, or one of its high performance drop-in replacements (such as the multi-threaded Intel MKL, or AMD ACML, or OpenBLAS libraries)
- A delayed evaluation approach is employed (at compile-time) to combine several operations into one and reduce (or eliminate) the need for temporaries; this is automatically accomplished through template meta-programming
- 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
- The library is open-source software, and is distributed under a license that is useful in both open-source and commercial/proprietary contexts
- Primarily developed at NICTA (Australia) by Conrad Sanderson, with contributions from around the world
- **Download latest version**

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.

Armadillo Highlights

- 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).
- Modern code, building upon and extending from earlier matrix libraries.
- Responsive and active maintainer, frequent updates.
- Used eg by MLPACK, see Curtin et al (JMLR, 2013)

RcppArmadillo Highlights

- 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

Example: Eigen values

```cpp
#include <RcppArmadillo.h>

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

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

Example: Eigen values

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

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

```r
eigen(M)$values
```

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

Example: Vector products

```
#include <RcppArmadillo.h>

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

// 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;
}
```

`FastLm` Case Study: Faster Linear Model with FastLm

Background

- Implementations of `fastLm()` have been a staple during development of Rcpp
- First version was in response to a question by Ivo Welch 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

```cpp
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:

```cpp
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@don:~$ Rscript ~/git/rcpparmadillo/inst/examples/fastLm.r
                       test replications relative elapsed
3          fLmConstRef(X, y)         5000    1.000   0.245
2          fLmTwoCasts(X, y)         5000    1.045   0.256
4              fLmSEXP(X, y)         5000    1.094   0.268
1           fLmOneCast(X, y)         5000    1.098   0.269
6   fastLmPureDotCall(X, y)          5000    1.118   0.274
8               lm.fit(X, y)         5000    1.673   0.410
5           fastLmPure(X, y)         5000    1.763   0.432
7 fastLm(frm, data = trees)          5000   30.612   7.500
9     lm(frm, data = trees)          5000   30.796   7.545
## continued below
```

Benchmark

```
## continued from above
                     test replications relative elapsed
2       fLmTwoCasts(X, y)        50000    1.000   2.327
3           fLmSEXP(X, y)        50000    1.049   2.442
4       fLmConstRef(X, y)        50000    1.050   2.444
1        fLmOneCast(X, y)        50000    1.150   2.677
6 fastLmPureDotCall(X, y)        50000    1.342   3.123
5        fastLmPure(X, y)        50000    1.988   4.627
7            lm.fit(X, y)        50000    2.141   4.982
edd@don:~$
```

VAR(1) Case Study

Simulating a VAR(1) system of k variables:

$$X_t = X_{t-1}B + E_t$$

where X_t is a row vector of length k, B is a k by k matrix and E_t is a row of the error matrix of k columns.

We use $k = 2$ for this example.

VAR(1) IN R

```r
## parameter and error terms used throughout
a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),ncol=2)

## Let's start with the R version
rSim <- function(coeff, errors) {
    simdata <- matrix(0, nrow(errors), ncol(errors))
    for (row in 2:nrow(errors)) {
        simdata[row,] = coeff %*% simdata[(row-1),] +
            errors[row,]
    }
    return(simdata)
}
rData <- rSim(a, e)                       # generated by R
```

VAR(1) IN C++

```
arma::mat rcppSim(const arma::mat& coeff,
                  const arma::mat& errors) {
    int m = errors.n_rows;
    int n = errors.n_cols;
    arma::mat simdata(m,n);
    simdata.row(0) = arma::zeros<arma::mat>(1,n);
    for (int row=1; row<m; row++) {
        simdata.row(row) = simdata.row(row-1) * coeff +
            errors.row(row);
    }
    return simdata;
}
```

Benchmark

```
Rcpp::sourceCpp("code/arma_var1.cpp")
rbenchmark::benchmark(rSim(a,e), rcppSim(a, e))[,1:4]
```

```
##             test replications elapsed relative
## 2 rcppSim(a, e)          100   0.018    1.000
## 1    rSim(a, e)          100   0.888   49.333
```

Kalman Filter Case Study

The position of an object is estimated based on past values of 6×1 state vectors X and Y for position, V_X and V_Y for speed, and A_X and A_Y for acceleration.

Position updates as a function of the speed

$$X = X_0 + V_X dt \quad \text{and} \quad Y = Y_0 + V_Y dt,$$

which is updated as a function of the (unobserved) acceleration:

$$V_x = V_{X,0} + A_X dt \quad \text{and} \quad V_y = V_{Y,0} + A_Y dt.$$

MATLAB CODE: kalmanfilter.m

```
% Copyright 2010 The MathWorks, Inc.
function y = kalmanfilter(z)
  dt=1;
  % Initialize state transition matrix
  A=[ 1 0 dt 0 0 0; 0 1 0 dt 0 0;...  % [x ], [y ]
      0 0 1 0 dt 0; 0 0 0 1 0 dt;...  % [Vx], [Vy]
      0 0 0 0 1 0 ; 0 0 0 0 0 1 ];    % [Ax], [Ay]
  H = [ 1 0 0 0 0 0; 0 1 0 0 0 0 ];   % Init. measuremnt mat
  Q = eye(6);
  R = 1000 * eye(2);
  persistent x_est p_est              % Init. state cond.
  if isempty(x_est)
    x_est = zeros(6, 1);              % x_est=[x,y,Vx,Vy,Ax,Ay]'
    p_est = zeros(6, 6);
  end

  x_prd = A * x_est;                  % Predicted state and covariance
  p_prd = A * p_est * A' + Q;

  S = H * p_prd' * H' + R;            % Estimation
  B = H * p_prd';
  klm_gain = (S \ B)';

  % Estimated state and covariance
  x_est = x_prd + klm_gain * (z - H * x_prd);
  p_est = p_prd - klm_gain * H * p_prd;
  y = H * x_est;                      % Compute the estimated measurements
end                                   % of the function
```

MATLAB CODE: kalmanM.m WITH LOOP

```
function Y = kalmanM(pos)
  dt=1;
  %% Initialize state transition matrix
  A=[ 1 0 dt 0 0 0;...     % [x  ]
      0 1 0 dt 0 0;...     % [y  ]
      0 0 1 0 dt 0;...     % [Vx]
      0 0 0 1 0 dt;...     % [Vy]
      0 0 0 0 1 0 ;...     % [Ax]
      0 0 0 0 0 1 ];       % [Ay]
  H = [ 1 0 0 0 0 0; 0 1 0 0 0 0 ];    % Initialize measurement matrix
  Q = eye(6);
  R = 1000 * eye(2);
  x_est = zeros(6, 1);                 % x_est=[x,y,Vx,Vy,Ax,Ay]'
  p_est = zeros(6, 6);
  numPts = size(pos,1);
  Y = zeros(numPts, 2);
  for idx = 1:numPts
    z = pos(idx, :)';
    x_prd = A * x_est;                 % Predicted state and covariance
    p_prd = A * p_est * A' + Q;
    S = H * p_prd' * H' + R;           % Estimation
    B = H * p_prd';
    klm_gain = (S \ B)';
    x_est = x_prd + klm_gain * (z - H * x_prd);  % Estimated state and covariance
    p_est = p_prd - klm_gain * H * p_prd;
    Y(idx, :) = H * x_est;             % Compute the estimated measurements
  end
end   % of the function
```

Now in R

```r
FirstKalmanR <- function(pos) {
    kalmanfilter <- function(z) {
        dt <- 1
        A <- matrix(c( 1, 0, dt, 0, 0, 0, 0, 1, 0, dt, 0, 0,  # x,  y
                       0, 0, 1, 0, dt, 0, 0, 0, 0, 1, 0, dt,  # Vx, Vy
                       0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0,  1), # Ax, Ay
                    6, 6, byrow=TRUE)
        H <- matrix( c(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),
                    2, 6, byrow=TRUE)
        Q <- diag(6)
        R <- 1000 * diag(2)
        xprd <- A %*% xest          # predicted state and covriance
        pprd <- A %*% pest %*% t(A) + Q
        S <- H %*% t(pprd) %*% t(H) + R     # estimation
        B <- H %*% t(pprd)
        kalmangain <- t(solve(S, B))
        ## estimated state and covariance, assign to vars in parent env
        xest <<- xprd + kalmangain %*% (z - H %*% xprd)
        pest <<- pprd - kalmangain %*% H %*% pprd
        y <- H %*% xest                    # compute the estimated measurements
    }
    xest <- matrix(0, 6, 1)
    pest <- matrix(0, 6, 6)
    N <- nrow(pos)
    y <- matrix(NA, N, 2)
    for (i in 1:N) y[i,] <- kalmanfilter(t(pos[i,,drop=FALSE]))
    invisible(y)
}
```

Improved in R

```r
KalmanR <- function(pos) {
    kalmanfilter <- function(z) {
        xprd <- A %*% xest                        # predicted state and covariance
        pprd <- A %*% pest %*% t(A) + Q
        S <- H %*% t(pprd) %*% t(H) + R           # estimation
        B <- H %*% t(pprd)
        kalmangain <- t(solve(S, B))
        xest <<- xprd + kalmangain %*% (z - H %*% xprd)    # est. state and covariance
        pest <<- pprd - kalmangain %*% H %*% pprd          # ass. to vars in parent env
        y <- H %*% xest                           # compute the estimated measurements
    }
    dt <- 1
    A <- matrix( c( 1, 0, dt, 0, 0, 0,  # x
                   0, 1, 0, dt, 0, 0,   # y
                   0, 0, 1, 0, dt, 0,   # Vx
                   0, 0, 0, 1, 0, dt,   # Vy
                   0, 0, 0, 0, 1,  0,   # Ax
                   0, 0, 0, 0, 0,  1),  # Ay
                6, 6, byrow=TRUE)
    H <- matrix( c(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0), 2, 6, byrow=TRUE)
    Q <- diag(6)
    R <- 1000 * diag(2)
    N <- nrow(pos)
    Y <- matrix(NA, N, 2)
    xest <- matrix(0, 6, 1)
    pest <- matrix(0, 6, 6)
    for (i in 1:N) Y[i,] <- kalmanfilter(t(pos[i,,drop=FALSE]))
    invisible(Y)
}
```

And now in C++

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

#include <RcppArmadillo.h>

using namespace arma;

class Kalman {
private:
    mat A, H, Q, R, xest, pest;
    double dt;

public:
    // constructor, sets up data structures
    Kalman() : dt(1.0) {
        A.eye(6,6);
        A(0,2) = A(1,3) = A(2,4) = A(3,5) = dt;
        H.zeros(2,6);
        H(0,0) = H(1,1) = 1.0;
        Q.eye(6,6);
        R = 1000 * eye(2,2);
        xest.zeros(6,1);
        pest.zeros(6,6);
    }

   // cont. below
```

And now in C++

```
// continued
    // sole member function: estimate model
    mat estimate(const mat & Z) {
       unsigned int n = Z.n_rows, k = Z.n_cols;
       mat Y = zeros(n, k);
       mat xprd, pprd, S, B, kalmangain;
       colvec z, y;

       for (unsigned int i = 0; i<n; i++) {
           z = Z.row(i).t();
           // predicted state and covariance
           xprd = A * xest;
           pprd = A * pest * A.t() + Q;
           // estimation
           S = H * pprd.t() * H.t() + R;
           B = H * pprd.t();
           kalmangain = (solve(S, B)).t();
           // estimated state and covariance
           xest = xprd + kalmangain * (z - H * xprd);
           pest = pprd - kalmangain * H * pprd;
           // compute the estimated measurements
           y = H * xest;
           Y.row(i) = y.t();
       }
       return Y;
    }
};
```

And now in C++

And the call:

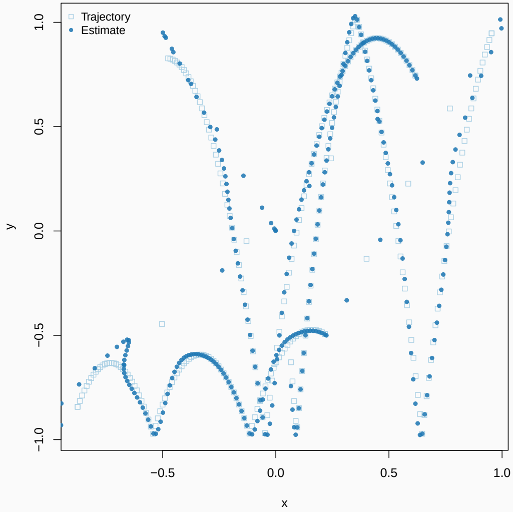
```cpp
// [[Rcpp::export]]
mat KalmanCpp(mat Z) {
  Kalman K;
  mat Y = K.estimate(Z);
  return Y;
}
```

Benchmark

```r
library(rbenchmark)
Rcpp::sourceCpp("code/kalman.cpp")
source("code/kalman.R")
p <- as.matrix(read.table("code/pos.txt",
                          header=FALSE,
                          col.names=c("x","y")))
benchmark(KalmanR(p), FirstKalmanR(p), KalmanCpp(p),
          order="relative", replications=500)[,1:4]
```

```
##              test replications elapsed relative
## 3    KalmanCpp(p)          500   3.910    1.000
## 1      KalmanR(p)          500  17.999    4.603
## 2 FirstKalmanR(p)          500  18.270    4.673
```

Reproduced Figure



Sparse Matrix Case Study

A nice example for work on R objects.

```
library(Matrix)
i <- c(1,3:8)
j <- c(2,9,6:10)
x <- 7 * (1:7)
A <- sparseMatrix(i, j, x = x)
A
```

```
## 8 x 10 sparse Matrix of class "dgCMatrix"
##
## [1,] . 7 . . .  .  .  .  .  .
## [2,] . . . . .  .  .  .  .  .
## [3,] . . . . .  .  .  . 14  .
## [4,] . . . . . 21  .  .  .  .
## [5,] . . . . .  . 28  .  .  .
## [6,] . . . . .  .  . 35  .  .
## [7,] . . . . .  .  .  . 42  .
## [8,] . . . . .  .  .  .  . 49
```

Sparse Matrix

```r
str(A)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:7] 0 3 4 5 2 6 7
##   ..@ p       : int [1:11] 0 0 1 1 1 1 2 3 4 6 ...
##   ..@ Dim     : int [1:2] 8 10
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:7] 7 21 28 35 14 42 49
##   ..@ factors : list()
```

Note how the construction was in terms of $< i, j, x >$, yet the representation in in terms of $< i, p, x >$ – CSC format.

Sparse Matrix

```cpp
#include <RcppArmadillo.h>

using namespace Rcpp;
using namespace arma;

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

// [[Rcpp::export]]
sp_mat armaEx(S4 mat, bool show) {
    IntegerVector dims = mat.slot("Dim");
    arma::urowvec i = Rcpp::as<arma::urowvec>(mat.slot("i"));
    arma::urowvec p = Rcpp::as<arma::urowvec>(mat.slot("p"));
    arma::vec x     = Rcpp::as<arma::vec>(mat.slot("x"));

    int nrow = dims[0], ncol = dims[1];
    arma::sp_mat res(i, p, x, nrow, ncol);
    if (show) Rcpp::Rcout << res << std::endl;
    return res;
}
```

Sparse Matrix

```
Rcpp::sourceCpp('code/arma_sparse.cpp')
B <- armaEx(A, TRUE)
```

```
## [matrix size: 8x10; n_nonzero: 7; density: 8.75%]
##
##      (0, 1)          7.0000
##      (3, 5)         21.0000
##      (4, 6)         28.0000
##      (5, 7)         35.0000
##      (2, 8)         14.0000
##      (6, 8)         42.0000
##      (7, 9)         49.0000
```

RcppMLPACK

Machine Learning

Overview

Among the 1000+ CRAN packages using Rcpp, several wrap Machine Learning libraries.

Here are three:

- RcppShark based on Shark
- RcppMLPACK based on MLPACK
- dlib based on DLib

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
- “Simple” packaging by embedding, copy of MLPACK now stale

MLPACK ‘v2’

High-level:

- A few of us are trying to update RcppMLPACK to ‘v2’
- Instead of embedding, amd external library is used
- This makes deployment a little tricker on Windows and macOS
- MLPACK uses Boost, this created issues with (older) RStudio builds

MLPACK

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

```cpp
#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);
}
```

RcppMLPACK: K-Means Example

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

```
#include <RcppMLPACK.h>             // MLPACK, Rcpp and RcppArmadillo

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

// [[Rcpp::export]]
Rcpp::List 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, lambda, intercept);
    arma::vec parameters = lr.Parameters();
    arma::vec fittedValues(vecY.n_elem);
    lr.Predict(matX,  fittedValues);

    return Rcpp::List::create(Rcpp::Named("parameters") = parameters,
                              Rcpp::Named("fitted") = fittedValues);
}
```

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

all.equal(unname(fitted(lmfit)),  c(mlfit[["fitted"]]))
```

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

RcppMLPACK: Logistic Regression Example

```cpp
#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::vec 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

```r
suppressMessages(library(utils))
library(RcppMLPACK)
example(logisticRegression)
```

```
##
## lgstcR> data(trainSet)
##
## lgstcR> mat <- t(trainSet[, -5])     ## train data, transpose and removing class labels
##
## lgstcR> lab <- trainSet[, 5]         ## class labels for train set
##
## lgstcR> logisticRegression(mat, lab)
## $parameters
## [1] -11.0819909  13.9022481   0.8034972  -9.3485217 -13.0869968
##
##
## lgstcR> testMat <- t(testSet[, -5])  ## test data
##
## lgstcR> logisticRegression(mat, lab, testMat)
## $parameters
## [1] -11.0819909  13.9022481   0.8034972  -9.3485217 -13.0869968
##
## $results
## [1] 0 0 0 1 1 1 1
```

RcppMLPACK: Nearest Neighbors Example

```cpp
#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

Questions?

Contact

http://dirk.eddelbuettel.com

dirk@eddelbuettel.com

@eddelbuettel