High-Performance Computing with Rcpp and RcppArmadillo

Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org
edd@debian.org
@eddelbuettel

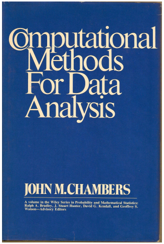
Big Data and Open Science with R
Warren Center for Network and Data Sciences
University of Pennsylvania, Philadelphia, PA
21 November 2014

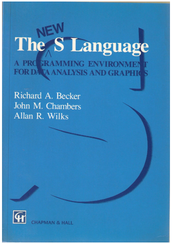
Outline

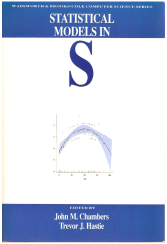
1. Why
   - R
   - C++
   - Vision
   - Features

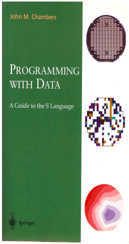
Why R?

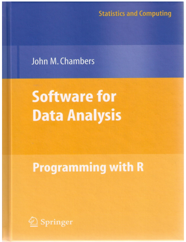
Programming with Data











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

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

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

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

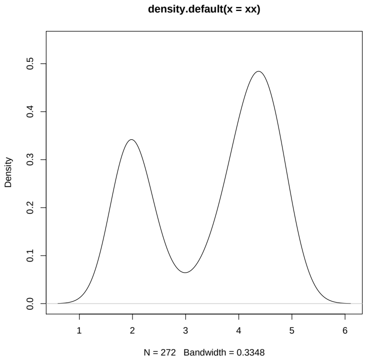
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

Thanks to John Chambers for sending me high-resolution scans of the covers of his books.

Why R?

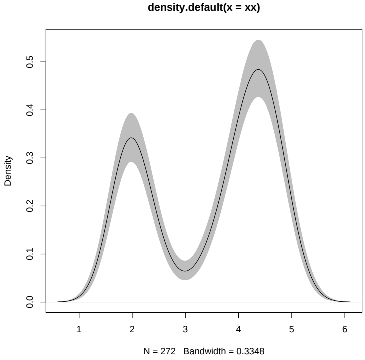
Succinct and expressive

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



Why R?
Succinct and expressive

```
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)
```



The example was posted by Greg Snow on r-help a few years ago.

Why R?

Interactive

R enables us to

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

making it a preferred environment for many data analysts.

Why R?

Extensible

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 S Urbanek
- **C++** but essentially at the bare-bones level of C

So 'in theory' this worked – yet tedious 'in practice'.

Why C++?

- Asking Google [currently] leads to about 42 million hits.
- Wikipedia: *C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language.*
- C++ is industrial-strength, vendor-independent, widely-used, and *still evolving*.
- In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API.
- As a widely used language it also has good tool support (debuggers, profilers, code analysis).

Why C++?

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

C provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.

Object-Oriented C++ 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 STL which is a specific template library which is powerful but has its own conventions.

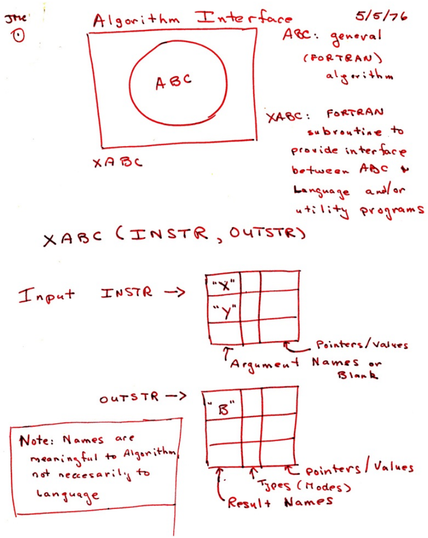
C++11 adds enough to be called a fifth language.

NB: Meyers original list of four language 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 a language between the machine level and C++”
- Yet also powerfully abtract and high-level
- C++11 and beyond are a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded

Interface Vision



Interface Vision

- Use trusted numerical libraries (mostly/exclusively written in Fortran)
- Provide environment which statistician could use more easily
- Enable interactive and iterative data exploration
- Make it extensibilty for research into statistical methods
- C.f. John Chambers (2008) regarding “Mission” and “Directive”

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)

Interpeted code with

- An accessible high-level language made for *Programming with Data*
- An interactive workflow for data analysis
- Support for rapid prototyping, research, and experimentation

Why Rcpp?

Easy to learn it really does not have to be that complicated – we will look at a few examples

Easy to use as it avoids build and OS system complexities thanks to the R infrastrucure

Expressive 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*

Outline

2 What

- R API
- C++

What can Rcpp do?

Everything evolves around `.Call`

At the C++ level:

```
SEXP foo(SEXP a, SEXP b, SEXP C, ...)
```

and at the R level:

```
res <- .Call("foo", a, b, c, ...,
             PACKAGE="mypkg")
```

What can Rcpp do?
Seamless interchange of R objects: C API of R

```
#include <R.h>
#include <Rdefines.h>
SEXP convolve2(SEXP a, SEXP b) {
    int i, j, na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    PROTECT(a = AS_NUMERIC(a));
    PROTECT(b = AS_NUMERIC(b));
    na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
    PROTECT(ab = NEW_NUMERIC(nab));
    xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
    xab = NUMERIC_POINTER(ab);
    for(i = 0; i < nab; i++) xab[i] = 0.0;
    for(i = 0; i < na; i++)
        for(j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return(ab);

}
```

What can Rcpp do?

Seamless interchange of R objects: Rcpp version

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector convolveCpp(NumericVector a, NumericVector b) {
    int na = a.size(), nb = b.size();
    int nab = na + nb - 1;
    NumericVector xab(nab);

    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            xab[i + j] += a[i] * b[j];

    return xab;
}
```

What can Rcpp do?

Seamless interchange of R objects

- Any R object can be passed down to C++ code: vectors, matrices, list, ...
- But also functions, environments and more.
- This includes S3 and S4 objects as well as Reference Classes.
- Object attributes can be accessed directly.
- Objects can be created at the C++ level, and the R garbage collector *does the right thing* as if were an R-created object.

What can Rcpp do?

Seamless use of RNGs

```
set.seed(42); runif(5)

## [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455

cppFunction('
NumericVector r1(int n) {
   NumericVector x(n);
   for (int i=0; i<n; i++) x[i] = R::runif(0,1);
   return(x);
}')
set.seed(42); r1(5)

## [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455

cppFunction('NumericVector r2(int n) { return runif(n,0,1); }')
set.seed(42); r2(5)

## [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455
```

What can Rcpp do?

Sugar: R version

```
piR <- function(N) {
    x <- runif(N)
    y <- runif(N)
    d <- sqrt(x^2 + y^2)
    return(4 * sum(d <= 1.0) / N)
}
```

What can Rcpp do?

Sugar: C++ version

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

// [[Rcpp::export]]
double piSugar(const int N) {
    NumericVector x = runif(N);
    NumericVector y = runif(N);
    NumericVector d = sqrt(x*x + y*y);
    return 4.0 * sum(d <= 1.0) / N;
}
```

Outline

3. When
   - A First Example
   - A Second Example
   - Numbers
   - Selection

When do we use Rcpp?

Easy speedup: An Introductory Example

Consider a function defined as

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

When do we use Rcpp?

Easy speedup: Simple R Implementation

```
fibR <- function(n) {
    if (n < 2) return(n)
    return(fibR(n-1) + fibR(n-2))
}
## Using it on first 11 arguments
sapply(0:10, fibR)

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

When do we use Rcpp?

Easy speedup: Timing R Implementation

```
benchmark(fibR(10),fibR(15),fibR(20))[,1:4]

##       test replications elapsed relative
## 1 fibR(10)          100   0.022    1.000
## 2 fibR(15)          100   0.221   10.045
## 3 fibR(20)          100   2.476  112.545
```

When do we use Rcpp?

Easy speedup: C++ Implementation

```
cppFunction("
  int fibCpp(int n) {
    if (n < 2) return(n);
    return(fibCpp(n-1) + fibCpp(n-2));
}")
## Using it on first 11 arguments
sapply(0:10, fibCpp)

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

When do we use Rcpp?

Easy speedup: Putting it all together

```
fibR <- function(n) {
    if (n<2) return(n)
    return(fibR(n-1) + fibR(n-2))
}
cppFunction('int fibCpp(int n) {
    if (n<2) return n;
    return fibCpp(n-2) + fibCpp(n-1);
}')
benchmark(fibR(25), fibCpp(25), order="relative")[,1:4]


##          test replications elapsed relative
## 2 fibCpp(25)          100   0.070    1.000
## 1   fibR(25)          100  27.597  394.243
```

When do we use Rcpp?

Easy speedup:: VAR(1) Simulation

Let's consider a simple possible VAR(1) system of k variables.

For $k = 2$:

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

where X_t is a row vector of length 2, B is a 2 by 2 matrix and E_t is a row of the error matrix of 2 columns.

When do we use Rcpp?

Easy speedup:: VAR(1) Simulation

In R code, given both the coefficient and error matrices (revealing *k* and *n*):

```
rSim <- function(B,E) {
    X <- matrix(0,nrow(E), ncol(E))
    for (r in 2:nrow(E)) {
        X[r,] = X[r-1, ] %*% B + E[r, ]
    }
    return(X)
}
```

When do we use Rcpp?

Easy speedup: VAR(1) Simulation

```
cppFunction('arma::mat cppSim(arma::mat B, arma::mat E)
    int m = E.n_rows; int n = E.n_cols;
    arma::mat X(m,n);
    X.row(0) = arma::zeros<arma::mat>(1,n);
    for (int r=1; r<m; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X; }', depends="RcppArmadillo")
a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),ncol=2)
benchmark(cppSim(a,e), rSim(a,e),
          order="relative")[,1:4]

##           test replications elapsed relative
## 1 cppSim(a, e)          100   0.029    1.000
## 2   rSim(a, e)          100   2.585   89.138
```

When do we use Rcpp?

New things: Easy access to C/C++ libraries

- Sometimes speed is not the only reason
- C and C++ provide a enormous amount of libraries and APIs we may want to use
- Easy to provide access to as **Rcpp** eases data transfer to/from R
- *Rcpp modules* can make it even easier

Where is Rcpp being used?

Numbers as of November 2014

Rcpp is

- used by 296 packages on CRAN
- used by another 40 package on BioConductor
- cited about 150 times (Google Scholar count for 2011 JSS paper and 2013 Springer book)

Where is Rcpp being used?

Several well-known packages

- **Amelia**: Gary King et al: Multiple Imputation; uses **Rcpp** and **RcppArmadillo**
- **forecast**: Rob Hyndman et al: (Automated) Time-series forecasting; uses **Rcpp** and **RcppArmadillo**
- **RStan**: Andrew Gelman et al: Bayesian models / MCMC
- **rugarch**: Alexios Ghalanos: Sophisticated financial models; using **Rcpp** and **RcppArmadillo**
- **lme4**: Doug Bates et al: Hierarchical/Mixed Linear Models; uses **Rcpp** and **RcppEigen**.
- **dplyr, bigviz, ...**: Hadley Wickham: Data munging; high-dim. visualization for 10-100 million obs.

Outline

Compiled not Interpreted

R is more flexible – lazy evaluation, conputing on the language, ...

C++ is compiled. Source code becomes object code.

Object code is linked into a binary excutable.

Binaries can also be linked with other libraries. This permits reuse.

Statically Typed

In R an expression determines the type of variable it is assigned to. This very flexible. The type can also change.

Statically typed languages require a *type declaration*. The assigned type cannot change.

Standard types are `int`, `double`, `std::string` are scalar.

There are *container* types wrapping them in vectors, list and more.

A better C

C++ improves upon / extends C. So it worth reviewing C basics:

loops `for` and `while` are similar to R

conditional `if / else` is similar to R, `switch` as well.

functions share similarities with R; function signature behaviour is different also reflecting types

pointers for memory management and variable passing; C++ improves greatly on C in both

Object Oriented

C++ is object-oriented, but is different from R's S3, S4, ReferenceClasses etc.

`struct` allows us to regroup variables.

`class` extends this by adding functions (called "methods"), and more.

Generic Programming

The Standard Template Library brought an important change to the language.

Functions like `push_back()`, `begin()`, `end()`, `size()` exist for different container with *guaranteed* performance bounds for each container type.

Containers like `vector`, `list`, `set`, ... can be changed dependending on the programming need.

This is further extended by a (large) set of standard algorithms and operations such as `find`, `tranform`, `accumulate`, ...

Algorithms and iterators can be applied to different data structures with minimal change.

Template Programming

As the language is statically typed, we need `sum(vector<int> x)` as well as `sum(vector<double> x)`.

As this gets tedious, templates permit to write code where we can abstract the type: `sum(T x)` which then gets instantiated with approriate vector types.

Template programming moves execution from the *run-time* to the *compile-time* making it also intriguing for performance tuning.

Template programming is one of the most difficult aspects of C++, and does not have to use it in applications yet can still deploy it from libraries.

Outline

5. How
    - Setup
    - evalCpp
    - cppFunction
    - sourceCpp
    - skeleton

How do we use Rcpp?

Uses only standard R tools to build packages

Depending on the platform, one needs

- Windows: the Rtools kit for Windows, properly installed – see CRAN, the Installation manual and many tutorials; the **installr** package may help
- OS X: the Xcode *command-line tools* (plus possibly the Fortran compiler) – see Simon's pages and/or r-sig-mac list
- Linux: generally just work out of the box

Several environments can be used to work with **Rcpp** – RStudio is very popular.

No additional requirements for Rcpp beyond *being able to compile R packages*.

How do we use Rcpp?

Easy to test

```
## evaluate a C++ expression, retrieve result
evalCpp("2 + 2")

## [1] 4

## a little fancier
evalCpp("std::numeric_limits<double>::max()")

## [1] 1.797693e+308

## create ad-hoc R function 'square'
cppFunction('int square(int x) { return x*x;}')
square(7L)

## [1] 49
```

How do we use Rcpp?

Basic Usage: `evalCpp`

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```
evalCpp("2 * M_PI")

## [1] 6.283185
```

How do we use Rcpp?

Basic Usage: `cppFunction()`

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

```
cppFunction("
    int useCpp11() {
        auto x = 10;
        return x;
}", plugins=c("cpp11"))
useCpp11()  # same identifier as C++ function

## [1] 10
```

How do we use Rcpp?

Basic Usage: `sourceCpp()`

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

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**).

We have also provided plugins for other compiler features. These allow to enable support for C++11 (and beyond), as well as for OpenMP.

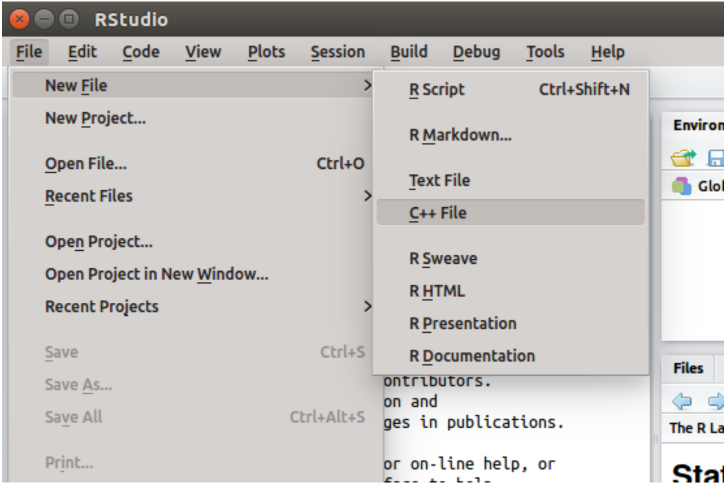
How do we use Rcpp?

Basic Usage: `Rcpp.package.skeleton()`

- To create a complete and working package, the `Rcpp.package.skeleton()` function can be used.
- It extends the base R function `package.skeleton()` and supports the same set of options.
- If installed, `pkgKitten::kitten()` is used to clean results of `Rcpp.package.skeleton()`.
- For **Rcpp** use is also supports (via additional options) *Rcpp Modules* and *Rcpp Attributes* both of which can be included with working examples
- The vignette Rcpp-package has complete details.

How do we use Rcpp?

RStudio makes it very easy: Single File



How do we use Rcpp?

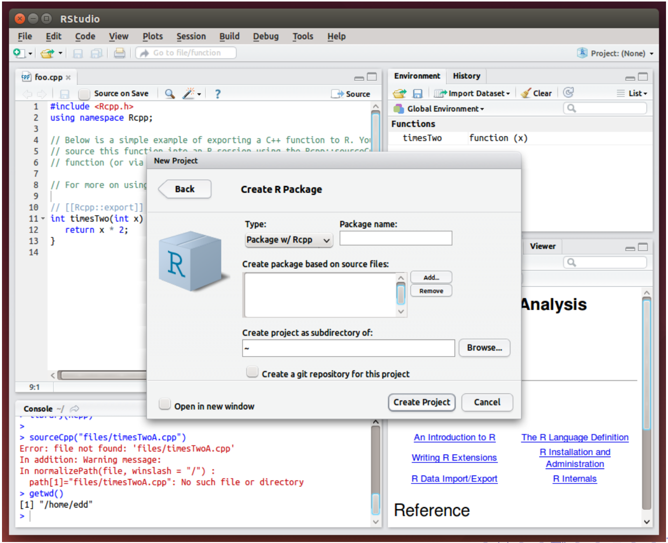
RStudio example cont'ed

The following file gets created:

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

// Below 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)

// For more on using Rcpp click the Help button on the editor
// toolbar

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

How do we use Rcpp?

RStudio makes it very easy: Package



Outline

6. Examples
    - CumSum
    - R Fun
    - Boost
    - Subset
    - xts
    - XPtr

Cumulative Sum

http://gallery.rcpp.org/articles/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){
    // initialize an accumulator variable
    double acc = 0;

    // initialize the result vector
    NumericVector res(x.size());

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

Cumulative Sum

http://gallery.rcpp.org/articles/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

http://gallery.rcpp.org/articles/vector-cumulative-sum/

Or just **Rcpp** sugar:

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

Of course, all results are the same.

```
cppFunction('NumericVector cumsum3(NumericVector x) {
                                   return cumsum(x); }')
x <- 1:10
all.equal(cumsum(x), cumsum3(x))

## [1] TRUE
```

Calling an R function from C++

http://gallery.rcpp.org/articles/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: Greatest Common Denominator

http://gallery.rcpp.org/articles/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);

}
```

Using Boost via BH: Lexical Cast

http://gallery.rcpp.org/articles/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
```

Using Boost via BH: Date Calculations

http://gallery.rcpp.org/articles/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);
}
```

Using Boost via BH: FOREACH

http://gallery.rcpp.org/articles/boost-foreach/

```
#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.

Using Boost via BH: Regular Expressions

http://gallery.rcpp.org/articles/boost-regular-expressions/

NB: Needs `Sys.setenv("PKG_LIBS"="-lboost_regex")` to link.

```
// 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 valid;

}
```

Vector Subsetting

http://gallery.rcpp.org/articles/subsetting/

New / improved in **Rcpp** 0.11.1:

```
#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];

}
```

Creating xts objects in C++

http://gallery.rcpp.org/articles/creating-xts-from-c++/

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

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

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

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

    return xv;
}
```

Function Pointers

http://gallery.rcpp.org/articles/passing-cpp-function-

Consider two simple functions modifying a given Armadillo vector:

```
// [[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);

}
```

Function Pointers

http://gallery.rcpp.org/articles/passing-cpp-function-

Using a `typedef` to declare an interface to a function taking and returning a vector — and 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);

}
```

Function Pointers

http://gallery.rcpp.org/articles/passing-cpp-function-

We then 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);
}
```

Function Pointers

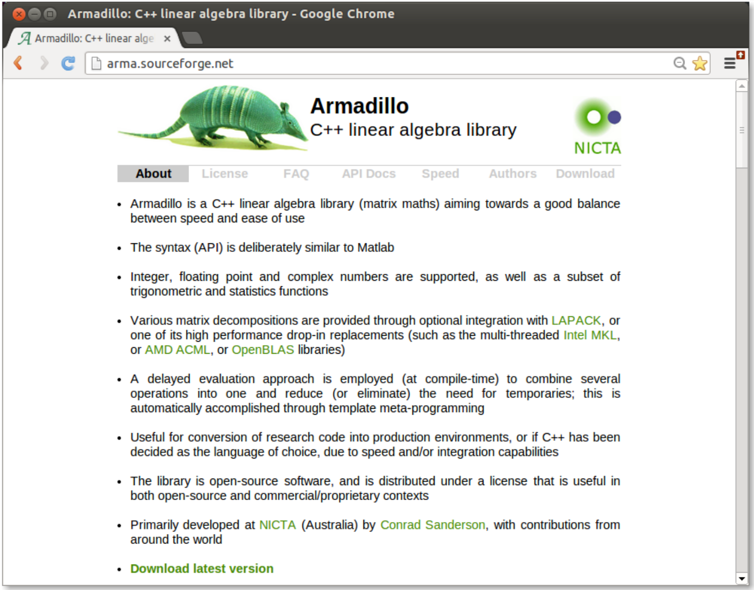
http://gallery.rcpp.org/articles/passing-cpp-function-

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

##      [,1]
## [1,]    2
## [2,]    4
## [3,]    6
## [4,]    8
```

Could use same mechanism for user-supplied functions, gradients, or samplers, ...

Outline

Armadillo



Armadillo: C++ linear algebra library - Google Chrome

arma.sourceforge.net

Armadillo

C++ linear algebra library

NICTA

About License FAQ API Docs Speed Authors Download

- 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?

From `arma.sf.net` and slightly edited

- 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.

What is Armadillo?

From `arma.sf.net` and slightly edited

- 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 and fields (3d) with all the common operations.
- Very good documentation and examples at website `http://arma.sf.net`, a technical report (Sanderson, 2010)
- Modern code, building upon and extending from earlier matrix libraries.
- Responsive and active maintainer, frequent updates.
- Used by MLPACK; cf 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 with R packages: just add `LinkingTo: RcppArmadillo, Rcpp` to DESCRIPTION (*i.e.*, no added cost beyond **Rcpp**)
- Data exchange really seamless from R via **Rcpp**
- Frequently updated; documentation includes Eddelbuettel and Sanderson (CSDA, 2014).

Well-know packages using RcppArmadillo

Amelia by Gary King et al: Multiple Imputation from cross-section, time-series or both;

forecast by Rob Hyndman et al: Time-series forecasting including state space and automated ARIMA modeling;

rugarch by Alexios Ghalanos: Sophisticated financial time series models;

gRbase by Søren Højsgaard: Graphical modeling

Armadillo Eigenvalues

http://gallery.rcpp.org/articles/armadillo-eigenvalues/

```
#include <RcppArmadillo.h>

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

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

Armadillo Eigenvalues

http://gallery.rcpp.org/articles/armadillo-eigenvalues/

```
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.
```

Multivariate Normal RNG Draw

http://gallery.rcpp.org/articles/simulate-multivariate-norn

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

// [[Rcpp::export]]
arma::mat mvrnormArma(int n, arma::vec mu,
                      arma::mat sigma) {
   arma::mat Y = arma::randn(n, sigma.n_cols);
   return arma::repmat(mu, 1, n).t() +
                 Y * arma::chol(sigma);
}
```

Faster Linear Model with FastLm

Background

- Implementations of ‘fastLm()‘ have been a staple all along the development of **Rcpp**
- The very first version was in response to a question by Ivo Welch on r-help.
- The 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 had since been reimplemented for **RcppArmadillo** and **RcppEigen**.

Faster Linear Model with FastLm

Initial RcppArmadillo `src/fastLm.cpp`

```
#include <RcppArmadillo.h>

extern "C" SEXP fastLm(SEXP Xs, SEXP ys) {

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

    arma::colvec coef = arma::solve(X, y);         // fit model y ∼ X
    arma::colvec res  = y - X*coef;                // residuals
    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

}
```

Faster Linear Model with FastLm

Edited version of earlier RcppArmadillo's `src/fastLm.cpp`

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

Faster Linear Model with FastLm

Current version of RcppArmadillo's `src/fastLm.cpp`

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

// [[Rcpp::export]]
List fastLm(const arma::mat& X, const arma::colvec& y) {
   int n = X.n_rows, k = X.n_cols;

   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    );

}
```

Faster Linear Model with FastLm

Note on as<>() casting with Armadillo

```
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);
```

Preferable if performance is a concern. Since last fall **RcppArmadillo** has efficient `const references` too.

Faster Linear Model with FastLm

Performance comparison

Running the script included in the **RcppArmadillo** package:

```
edd@max:~/svn/rcpp/pkg/RcppArmadillo/inst/examples$ r fastLm.r
Loading required package: Rcpp
                       test replications relative elapsed
2          fLmTwoCasts(X, y)         5000    1.000   0.188
3          fLmConstRef(X, y)         5000    1.000   0.188
1           fLmOneCast(X, y)         5000    1.005   0.189
5   fastLmPureDotCall(X, y)         5000    1.064   0.200
4           fastLmPure(X, y)         5000    2.000   0.376
7               lm.fit(X, y)         5000    2.691   0.506
6 fastLm(frm, data = trees)         5000   35.596   6.692
8     lm(frm, data = trees)         5000   44.883   8.438
edd@max:~/svn/rcpp/pkg/RcppArmadillo/inst/examples$
```

Kalman Filter

Setup at Mathworks site

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

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

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

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

Kalman Filter

Basic Matlab Function

```
% Copyright 2010 The MathWorks, Inc.
function y = kalmanfilter(z)
% #codegen
    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 ];
    Q = eye(6);
    R = 1000 * eye(2);
    persistent x_est p_est
    if isempty(x_est)
        x_est = zeros(6, 1);
        p_est = zeros(6, 6);
    end
        % Predicted state and covariance
        x_prd = A * x_est;
        p_prd = A * p_est * A' + Q;
        % Estimation
        S = H * p_prd' * H' + R;
        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;
        % Compute the estimated measurements
        y = H * x_est;
end                   % of the function
```

Plus a simple wrapper function calling this function.

Kalman Filter: In R

Easy enough – first naive solution

```r
FirstKalmanR <- function(pos) {

  kf <- function(z) {
    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)

    ## predicted state and covriance
    xprd <- A %*% xest
    pprd <- A %*% pest %*% t(A) + Q

    ## estimation
    S <- H %*% t(pprd) %*% t(H) + R
    B <- H %*% t(pprd)
    ##  kalmangain <- (S \ B)'
    kg <- t(solve(S, B))

    ## est. state and cov, assign to vars in parent env
    xest <<- xprd + kg %*% (z-H%*%xprd)
    pest <<- pprd - kg %*% H %*% pprd

    ## compute the estimated measurements
    y <- H %*% xest
  }

  xest <- matrix(0, 6, 1)
  pest <- matrix(0, 6, 6)

  for (i in 1:N) {
      y[i,] <- kf(t(pos[i,,drop=FALSE]))
  }

  invisible(y)
}
```

Kalman Filter: In R

Easy enough – with some minor refactoring

```r
KalmanR <- function(pos) {

  kf <- function(z) {
    ## predicted state and covriance
    xprd <- A %*% xest
    pprd <- A %*% pest %*% t(A) + Q

    ## estimation
    S <- H %*% t(pprd) %*% t(H) + R
    B <- H %*% t(pprd)
    ##   kg <- (S \ B)'
    kg <- t(solve(S, B))

    ## estimated state and covariance
    ## assigned to vars in parent env
    xest <<- xprd + kg %*% (z-H%*%xprd)
    pest <<- pprd - kg %*% H %*% pprd

    ## compute the estimated measurements
    y <- H %*% xest
  }
  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,] <- kf(t(pos[i,,drop=FALSE]))
  }
  invisible(y)
}
```

Kalman Filter: In C++

Using a simple class

```
// [[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) = dt;
    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);

  }
```

```
  // sole member func.: 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, kg;
    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();
      kg = (solve(S, B)).t();
      // estimated state and covariance
      xest = xprd + kg * (z - H * xprd);
      pest = pprd - kg * H * pprd;
      // compute estimated measurements
      y = H * xest;
      Y.row(i) = y.t();
    }
    return Y;
  }
```

Kalman Filter in C++

Trivial to use from R

Given the code from the previous slide, we just add

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

Kalman Filter: Performance

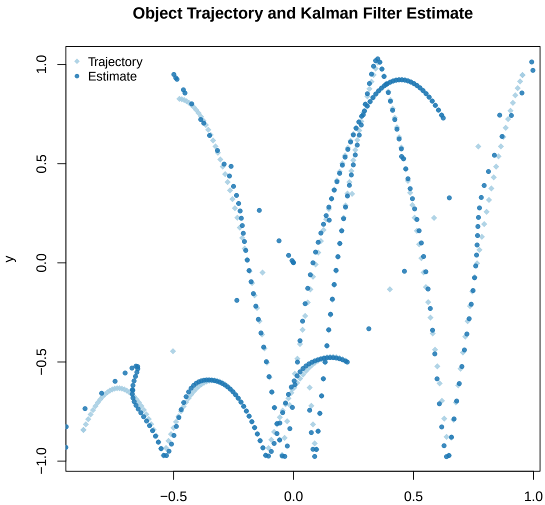
Quite satisfactory relative to R

Even byte-compiled 'better' R version is 66 times slower:

```
R> FirstKalmanRC <- cmpfun(FirstKalmanR)
R> KalmanRC <- cmpfun(KalmanR)
R>
R> stopifnot(identical(KalmanR(pos), KalmanRC(pos)),
+            all.equal(KalmanR(pos), KalmanCpp(pos)),
+            identical(FirstKalmanR(pos), FirstKalmanRC(pos)),
+            all.equal(KalmanR(pos), FirstKalmanR(pos)))
R>
R> res <- benchmark(KalmanR(pos), KalmanRC(pos),
+                  FirstKalmanR(pos), FirstKalmanRC(pos),
+                  KalmanCpp(pos),
+                  columns = c("test", "replications",
+                              "elapsed", "relative"),
+                  order="relative",
+                  replications=100)
R>
R> print(res)
                test replications elapsed relative
5     KalmanCpp(pos)          100   0.087   1.0000
2      KalmanRC(pos)          100   5.774  66.3678
1       KalmanR(pos)          100   6.448  74.1149
4 FirstKalmanRC(pos)          100   8.153  93.7126
3  FirstKalmanR(pos)          100   8.901 102.3103
```

Kalman Filter: Figure

Last but not least we can redo the plot as well



Outline

8 HPC
- Big Memory
- Parallel

Using the bigmemory package from Rcpp

http://gallery.rcpp.org/articles/using-bigmemory-with-rcpp/

Bigmemory is a wonderful package by Jay Emerson and Michael Kane.

It permits you to use with *large* objects outside of R's memory.

This can be useful for a single "chunk" of data access by several processes (or threads, for advanced users) sharing a "handle" to the data.

The Rcpp Gallery post by Mike Kane and Scott Ritchie gives you a full example; but it is a little too long to fit on one slide, and too advanced for our purposes.

RcppParallel

Fairly recent package by JJ

Parallel programming is hard.

Parallel programming is also hardware and OS-dependent.

A recent package by JJ tries to tackle both aspects.

It builds on top of the Intel Threading Building Blocks (where available) and the TinyThread (as a fallback).

The package comes with several examples, and the Rcpp Gallery has examples too. Discussing this in detail here is beyond the scope for today.

Outline

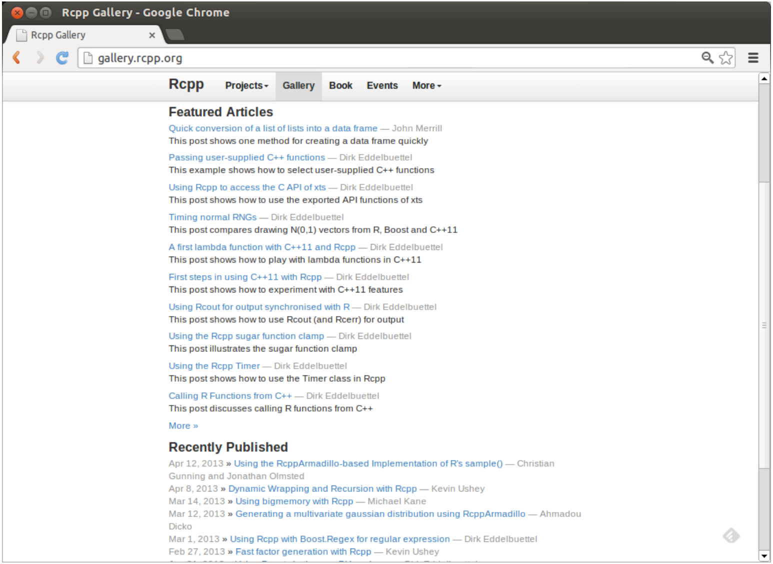
9 Doc
- Basics
- Gallery
- Book

What Else?
Basic Documentation

- The package comes with **eight pdf vignettes**, and numerous help pages.
- The introductory vignettes are now **published** (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp. Stat.& Data Anal.*).
- The **rcpp-devel** list is *the* recommended resource, generally very helpful, and fairly low volume.
- **StackOverflow** has over 500 posts too.
- Several blog posts introduce/discuss features.

What Else?

Rcpp Gallery: 80+ working and detailed examples



Rcpp Gallery - Google Chrome

Rcpp Gallery

gallery.rcpp.org

Rcpp Projects Gallery Book Events More

Featured Articles

Quick conversion of a list of lists into a data frame — John Merrill
This post shows one method for creating a data frame quickly

Passing user-supplied C++ functions — Dirk Eddelbuettel
This example shows how to select user-supplied C++ functions

Using Rcpp to access the C API of xts — Dirk Eddelbuettel
This post shows how to use the exported API functions of xts

Timing normal RNGs — Dirk Eddelbuettel
This post compares drawing N(0,1) vectors from R, Boost and C++11

A first lambda function with C++11 and Rcpp — Dirk Eddelbuettel
This post shows how to play with lambda functions in C++11

First steps in using C++11 with Rcpp — Dirk Eddelbuettel
This post shows how to experiment with C++11 features

Using Rcout for output synchronised with R — Dirk Eddelbuettel
This post shows how to use Rcout (and Rcerr) for output

Using the Rcpp sugar function clamp — Dirk Eddelbuettel
This post illustrates the sugar function clamp

Using the Rcpp Timer — Dirk Eddelbuettel
This post shows how to use the Timer class in Rcpp

Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

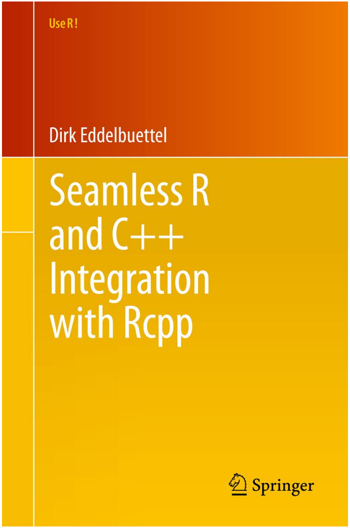
More »

Recently Published

Apr 12, 2013 » Using the RcppArmadillo-based Implementation of R's sample() — Christian Gunning and Jonathan Olmsted
Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey
Mar 14, 2013 » Using bigmemory with Rcpp — Michael Kane
Mar 12, 2013 » Generating a multivariate gaussian distribution using RcppArmadillo — Ahmadou Dicko
Mar 1, 2013 » Using Rcpp with Boost.Regex for regular expression — Dirk Eddelbuettel
Feb 27, 2013 » Fast factor generation with Rcpp — Kevin Ushey

What Else?

The Rcpp book



In print since June 2013