R and C++ Integration with Rcpp: Motivation and Examples

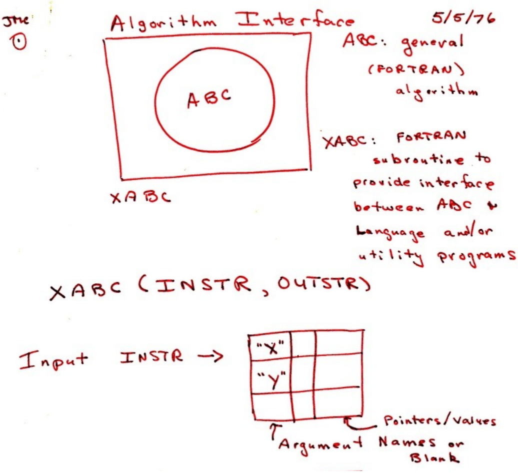
Dr. Dirk Eddelbuettel
edd@debian.org
dirk.eddelbuettel@R-Project.org

Guest Lecture on April 30, 2013
CMSC 12300 Computer Science with Applications-3
Department of Computer Science, University of Chicago

Outline

1. Introduction
2. Objects
3. Sugar
4. Usage
5. Examples
6. RInside
7. More

A “vision” from Bell Labs from 1976



JMC

Algorithm Interface 5/5/76

ABC: general (FORTRAN) algorithm

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

XABC (INSTR, OUTSTR)

Input INSTR →

Source: John Chambers’ talk at Stanford in October 2010; personal correspondence.

Passing any R object with ease: Sparse Matrix

See `http://gallery.rcpp.org/articles/armadillo-sparse-matrix/`

Define S4 object sparse matrix

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

resulting in

```
## 7 x 9 sparse Matrix of class "dgCMatrix"
##
## [1,] . 6 . . .  .  .  .  .
## [2,] . . . . .  .  .  .  .
## [3,] . . . . .  .  .  . 12
## [4,] . . . . . 18  .  .  .
## [5,] . . . . .  . 24  .  .
## [6,] . . . . .  .  . 30  .
## [7,] . . . . .  .  .  . 36
```

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

// [[Rcpp::export]]
void accessSparse(S4 mat) {
    IntegerVector dims = mat.slot("Dim");
    IntegerVector i = mat.slot("i");
    IntegerVector p = mat.slot("p");
    NumericVector x = mat.slot("x");

    int nrow = dims[0], ncol = dims[1];
    arma::sp_mat res(nrow, ncol);
    // ... some code
```

used via

```
library(Rcpp)
## compile/load/link example
sourceCpp("fileWitnExample.cpp")
## work on sparse matrix A
convertSparse(A)
```

A classic example

Consider a function defined as

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

A classic example: Simple R Implementation

R implementation:

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

A classic example: Running Simple R Implementation

Use:

```
f <- function(n) {
    if (n < 2) return(n)
    return(f(n-1) + f(n-2))
}
sapply(0:10, f)

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

A classic example: Timing Simple R Implementation

Timing:

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

##    test replications elapsed relative
## 1 f(10)          100   0.033     1.00
## 2 f(15)          100   0.379    11.48
## 3 f(20)          100   4.161   126.09
```

A classic example: A Simple C++ Implementation

```
int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2));
}
```

Deployed as:

```
library(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
```

A classic example: Comparing timing

Timing:

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

##    test replications elapsed relative
## 1 f(20)          100   4.103    586.1
## 2 g(20)          100   0.007      1.0
```

A nice 600-fold gain.

Well-know packages using Rcpp

Amelia	by Gary King et al: Multiple Imputation from cross-section, time-series or both; uses Rcpp and RcppArmadillo
forecast	by Rob Hyndman et al: Time-series forecasting including state space and automated ARIMA modeling; uses Rcpp and Armadillo
RStan	by Andrew Gelman et al: Rcpp helps with automatic model parsing / generation for MCMC / Bayesian modeling
rugarch	by Alexios Ghalanos: Sophisticated financial time series models using Rcpp and RcppArmadillo
bigviz	by Hadley Wickham: High-performance visualization of datasets in the 10-100 million observations range

Outline

1. Introduction
2. Objects
3. Sugar
4. Usage
5. Examples
6. RInside
7. More

Type mapping

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

```
library(Rcpp)
cppFunction("
  NumericVector logabs(NumericVector x) {
    return log(abs(x));
  }")
logabs(seq(-5, 5, by=2))

## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

Also note: vectorized C++!

Type mapping also with C++ STL types

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

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
   std::transform(x.begin(), x.end(), x.begin(),  f);
   return x;
}
```

And:

```
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))

## [1] 1.609 1.099 0.000 0.000 1.099 1.609
```

Type mapping is seamless

Simple outer product of a column vector (using Armadillo / RcppArmadillo):

```
cppFunction("arma::mat v(arma::colvec a) {return a*a.t();}",
depends="RcppArmadillo")
v(1:5)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    4    6    8   10
## [3,]    3    6    9   12   15
## [4,]    4    8   12   16   20
## [5,]    5   10   15   20   25
```

This uses implicit conversion via `as<>` and `wrap` – cf package vignette Rcpp-extending.

Outline

1 Introduction

2 Objects

3 Sugar

4 Usage

5 Examples

6 RInside

7 More

Syntactive 'sugar': Simulating π in R

Basic idea: for point (x, y), compute distance to origin. Do so repeatedly, and ratio of points below one to number N of simulations will approach $\pi/4$ as we fill the area of one quarter of the unit circle.

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

set.seed(5)
sapply(10^(3:6), piR)

## [1] 3.156 3.155 3.139 3.141
```

Syntactive ’sugar’: Simulating π in C++

The neat thing about Rcpp sugar enables us to write C++ code that looks almost as compact.

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

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

Apart from RNG set/reset, the code is essentially identical.

Syntactive 'sugar': Simulating π

And by using the same RNG, so are the results.

```
sourceCpp("code/piSugar.cpp")
set.seed(42); a <- piR(1.0e7)
set.seed(42); b <- piSugar(1.0e7)
identical(a,b)

## [1] TRUE

print(c(a,b), digits=7)

## [1] 3.140899 3.140899
```

Syntactive ’sugar’: Simulating π

The performance is close with a small gain for C++ as R is already vectorised:

```
library(rbenchmark)
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]

##             test replications elapsed relative
## 1      piR(1e+06)          100  12.980    1.725
## 2 piSugar(1e+06)          100   7.526    1.000
```

More about Sugar is in the package vignette Rcpp-sugar.

Outline

1 Introduction

2 Objects

3 Sugar

4 Usage

5 Examples

6 RInside

7 More

Basic Usage: `evalCpp`

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.
This allows us to quickly check C++ constructs.

```
evalCpp( "std::numeric_limits<double>::max()" )

## [1] 1.798e+308
```

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
```

Basic Usage: `sourceCpp()`

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail 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).

Basic Usage: Example using RcppArmadillo

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

#include <RcppArmadillo.h>

using namespace Rcpp;

// [[Rcpp::export]]
List fastLm(NumericVector yr, NumericMatrix Xr) {

   int n = Xr.nrow(), k = Xr.ncol();

   arma::mat X(Xr.begin(), n, k, false);
   arma::colvec y(yr.begin(), yr.size(), false);

   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 stderrest = arma::sqrt(
       sig2 * arma::diagvec( arma::inv(arma::trans(X)*X)) );

   return List::create(Named("coefficients") = coef,
                       Named("stderr")       = stderrest);
}
```

Basic Usage: Packages

Package are *the* standard unit of R code organization.

Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette Rcpp-package has fuller details.

As of April 2013, there are 110 packages on CRAN which use Rcpp, and a further 10 on BioConductor — with working, tested, and reviewed examples.

Outline

1. Introduction
2. Objects
3. Sugar
4. Usage
5. Examples
6. RInside
7. More

Cumulative Sum

See 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;
}
```

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;
}
```

Or just sugar:

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

Sugar head and tail

See http://gallery.rcpp.org/articles/sugar-head-tail/

Three largest:

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

// [[Rcpp::export]]
NumericVector top_n(NumericVector y,
                    int n) {
    NumericVector x = clone(y);
    // sort x in ascending order
    std::sort(x.begin(), x.end());
    return tail(x, n);
}

/*** R
set.seed(42)
x <- rnorm(10)
x
top_n(x, 3)
*/
```

Three smallest:

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

// [[Rcpp::export]]
NumericVector bottom_n(NumericVector y,
                       int n){
    NumericVector x = clone(y);
    // sort x in ascending order
    std::sort(x.begin(), x.end());
    return head(x, n);
}

/*** R
bottom_n(x, 3)
*/
```

Armadillo subsetting

See http://gallery.rcpp.org/articles/armadillo-subsetting/

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

using namespace Rcpp ;

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

/*** R
M <- matrix(1:9, 3, 3)
M
matrixSubset(M)
*/
```

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

using namespace Rcpp ;

// [[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;
}

/*** R
matrixSubset2(M)
*/
```

Calling an R function from C++

See http://gallery.rcpp.org/articles/r-function-from-c++/

```
/*** R
set.seed(42)
x <- rnorm(1e5)
fivenum(x)
*/

#include <Rcpp.h>

using namespace Rcpp;

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

/*** R
callFunction(x, fivenum)
*/
```

```
options(width=40)
sourceCpp("code/r-from-cpp.cpp")


##
## > set.seed(42)
##
## > x <- rnorm(1e5)
##
## > fivenum(x)
## [1] -4.043276 -0.682384 -0.002066
## [4]  0.673325  4.328091
##
## > callFunction(x, fivenum)
## [1] -4.043276 -0.682384 -0.002066
## [4]  0.673325  4.328091
```

A simple C++ Lambda example

See http://gallery.rcpp.org/articles/simple-lambda-func-c++11/

```
#include <Rcpp.h>

using namespace Rcpp;

// Important: enable C++11 via plugin
// [[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;
}
```

An R example use:

```
sourceCpp("code/lambda.cpp")
x <- c(1,2,3,4)
transformEx(x)


## [1]  1  4  9 16
```

Using Boost via BH

See http://gallery.rcpp.org/articles/using-boost-with-bh/

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

// One include file from Boost
#include <boost/date_time/gregorian/gregorian_types.hpp>

using namespace boost::gregorian;

// [[Rcpp::export]]
Rcpp::Date getIMMDate(int mon, int year) {
    // compute third Wednesday of given month / year
    date d = nth_day_of_the_week_in_month(nth_day_of_the_week_in_month::third,
                                          Wednesday, mon).get_date(year);
    date::ymd_type ymd = d.year_month_day();
    return Rcpp::wrap(Rcpp::Date(ymd.year, ymd.month, ymd.day));
}
```

We can test this from R:

```r
sourceCpp("code/boost-bh.cpp")
getIMMDate(6, 2013)


## [1] "2013-06-19"
```

Outline

1. Introduction
2. Objects
3. Sugar
4. Usage
5. Examples
6. RInside
7. More

The first example

examples/standard/rinside_sample0.cpp

```
#include <RInside.h>                    // for the embedded R via RInside

int main(int argc, char *argv[]) {

    RInside R(argc, argv);              // create an embedded R instance

    R["txt"] = "Hello, world!\n";       // assign a char* (string) to 'txt'

    R.parseEvalQ("cat(txt)");           // eval the init string, ignoring any returns

    exit(0);
}
```

Assign a variable, evaluate an expression—easy!

RInside in a nutshell

Key aspects:

- RInside uses the embedding API of R
- An instance of R is launched by the RInside constructor
- It behaves just like a regular R process
- We submit commands as C++ strings which are parsed and evaluated
- Rcpp is used to easily get data in and out from the enclosing C++ program.

Application example: Qt

RInside `examples/qt/`

The question is sometimes asked how to embed **RInside** in a larger program.

We have a nice example using **Qt**:

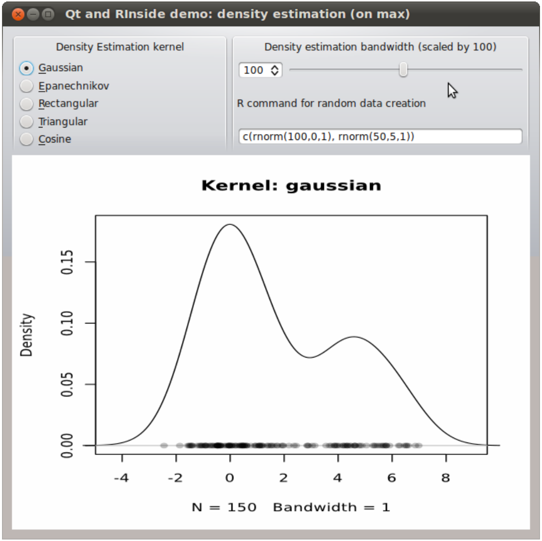
```
#include <QApplication>
#include "qtdensity.h"

int main(int argc, char *argv[]) {

    RInside R(argc, argv);         // embedded R inst.
    QApplication app(argc, argv);
    QtDensity qtdensity(R);        // passess by ref.
    return app.exec();
}
```

Application example: Qt density slider

RInside `examples/qt/`



Qt and RInside demo: density estimation (on max)

Density Estimation kernel

- ● Gaussian
- ○ Epanechnikov
- ○ Rectangular
- ○ Triangular
- ○ Cosine

Density estimation bandwidth (scaled by 100)

100

R command for random data creation

c(rnorm(100,0,1), rnorm(50,5,1))

This uses standard **Qt** / GUI paradigms of

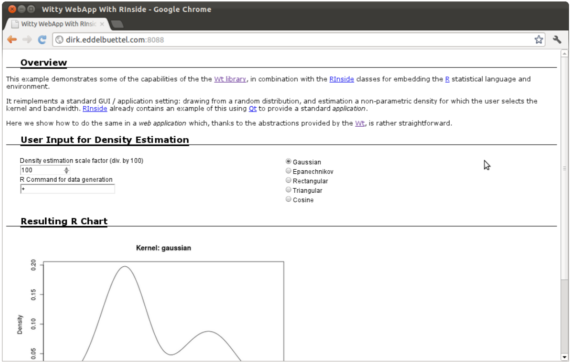
- radio buttons
- sliders
- textentry

all of which send values to the R process which provides a PNG image that is plotted.

Application example: Wt

RInside `examples/wt/`

Given the desktop application with **Qt**, the question arises how to deliver something similar “over the web” — and **Wt** helps.



Witty WebApp With RInside - Google Chrome
Witty WebApp With RInsic
dirk.eddelbuettel.com:8088
Kernel: gaussian
Density
0.05
0.10
0.15
0.20

Overview

This example demonstrates some of the capabilities of the the Wt library, in combination with the RInside classes for embedding the R statistical language and environment.

It reimplements a standard GUI / application setting: drawing from a random distribution, and estimation a non-parametric density for which the user selects the kernel and bandwidth. RInside already contains an example of this using Qt to provide a standard *application*.

Here we show how to do the same in a *web application* which, thanks to the abstractions provided by the Wt, is rather straightforward.

User Input for Density Estimation

Density estimation scale factor (div. by 100)
100

R Command for data generation
r

● Gaussian
○ Epanechnikov
○ Rectangular
○ Triangular
○ Cosine

Resulting R Chart

Wt is similar to **Qt** so the code needs only a few changes.

Wt takes care of all browser / app interactions and determines the most featureful deployment.

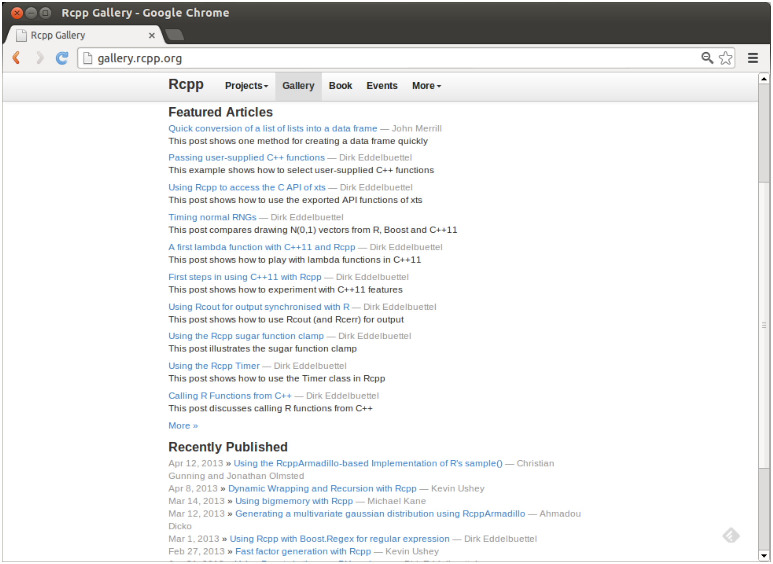
Outline

1. Introduction
2. Objects
3. Sugar
4. Usage
5. Examples
6. RInside
7. More

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.
- By now StackOverflow has a fair number of posts too.
- And a number of blog posts introduce/discuss features.

Rcpp Gallery



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

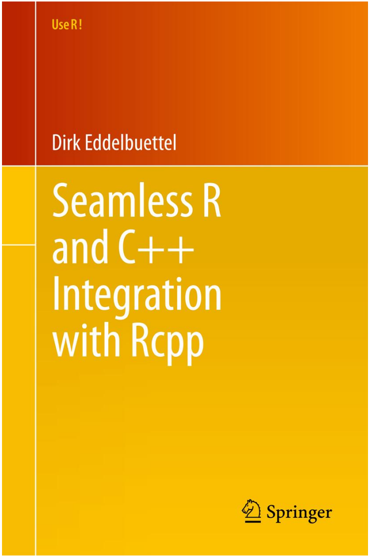
Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

More »

Recently Published

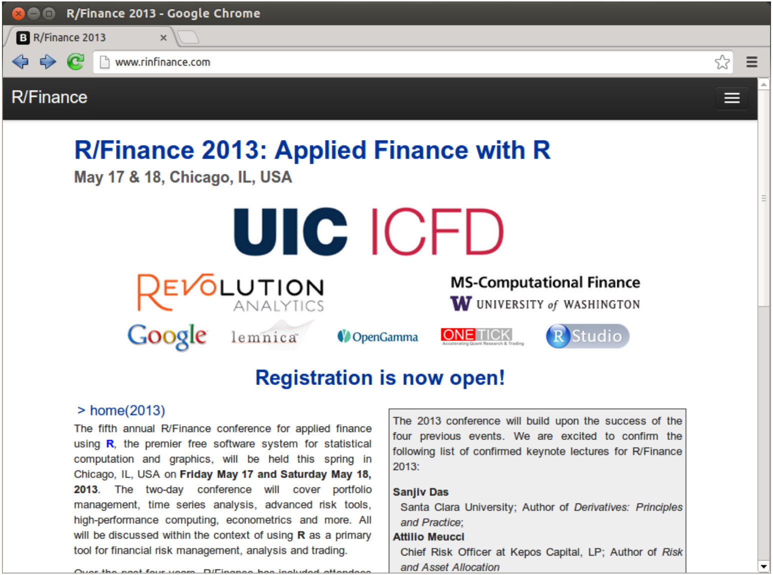
Apr 12, 2013 » Using the RcppArmadillo-based Implementation of R's sample() — Christian Gunning and Jonathan Olmsted
Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey
Mar 14, 2013 » Using bigmemory with Rcpp — Michael Kane
Mar 12, 2013 » Generating a multivariate gaussian distribution 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

The Rcpp book



Expected May 2013.
Real Soon Now.

Also: R/Finance in two weeks



R/Finance 2013: Applied Finance with R

May 17 & 18, Chicago, IL, USA

Registration is now open!

> home(2013)

The fifth annual R/Finance conference for applied finance using **R**, the premier free software system for statistical computation and graphics, will be held this spring in Chicago, IL, USA on **Friday May 17 and Saturday May 18, 2013**. The two-day conference will cover portfolio management, time series analysis, advanced risk tools, high-performance computing, econometrics and more. All will be discussed within the context of using **R** as a primary tool for financial risk management, analysis and trading.

Over the past four years, R/Finance has included attendees

The 2013 conference will build upon the success of the four previous events. We are excited to confirm the following list of confirmed keynote lectures for R/Finance 2013:

Sanjiv Das
Santa Clara University; Author of *Derivatives: Principles and Practice*;

Attilio Meucci
Chief Risk Officer at Kepos Capital, LP; Author of *Risk and Asset Allocation*