Rcpp by Examples

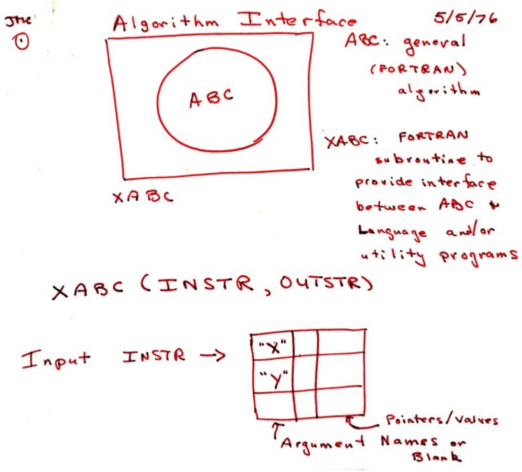
Dr Dirk Eddelbuettel
edd@debian.org
dirk.eddelbuettel@R-Project.org

Sydney Users of R Forum (suRf)
Sydney, Australia
10 July 2013

Outline

1. Introduction
2. Usage
3. Sugar
4. Examples from the Rcpp Gallery
5. RInside
6. More

A “vision” from Bell Labs from 1976



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

An Introductory Example

Consider a function defined as

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

An Introductory Example: Simple R Implementation

R implementation and use:

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

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

An Introductory Example: Timing R Implementation

Timing:

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

##     test replications elapsed relative
## 1 f(10)          100   0.034     1.00
## 2 f(15)          100   0.394    11.59
## 3 f(20)          100   4.384   128.94
```

An Introductory Example: 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)); }')
## Using it on first 11 arguments
sapply(0:10, g)

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

An Introductory Example: Comparing timing

Timing:

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

##    test replications elapsed relative
## 1 f(20)          100   4.286    612.3
## 2 g(20)          100   0.007      1.0
```

A nice 600-fold gain.

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

Use of `std::vector<double>` and STL algorithms:

```
#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;
}
```

Type mapping also with C++ STL types

Used via

```
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.

Basic Usage: FastLm using RcppArmadillo

A simple but speedy reimplementation of `lm()`

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

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 RcppArmadillo

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

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.

```r
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 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).

We are also starting to provide other compiler features via plugins. A first plugin to enable C++11 support was added in Rcpp 0.10.3.

Basic Usage: Packages

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

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

The vignette Rcpp-package has fuller details.

As of July 2013, there are 125 packages on CRAN which use Rcpp, and a further 10 on BioConductor — with working, tested, and reviewed examples.

Outline

1. Introduction
2. Usage
3. Sugar
4. Examples from the Rcpp Gallery
5. RInside
6. More

Syntactive 'sugar': Simulating π in R

Basic idea: for point (x, y), compute distance to origin. Do so repeatedly, and the 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.

```
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* is that it 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

Here, the performance gain is less dramatic as the R code is already vectorised:

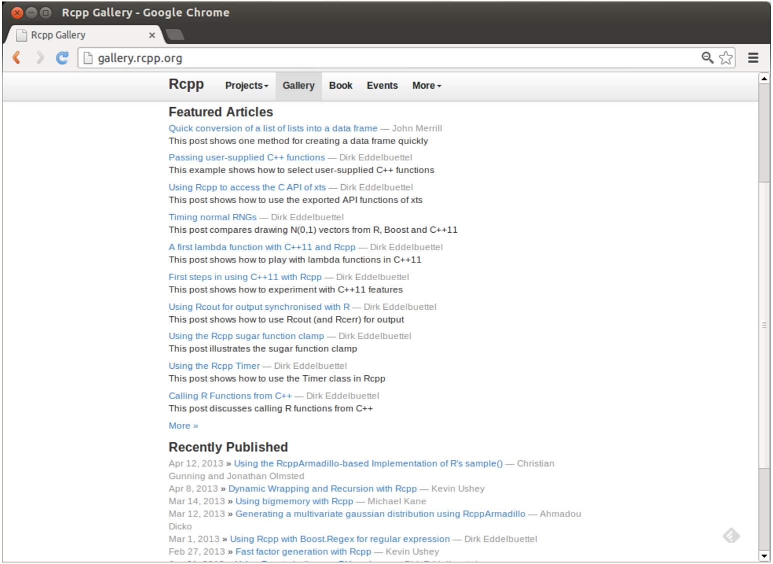
```
library(rbenchmark)
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]

##              test replications elapsed relative
## 1      piR(1e+06)          100  13.407    1.733
## 2 piSugar(1e+06)          100   7.738    1.000
```

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

Outline

From the 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

Cumulative Sum

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

A basic looped version:

```
#include <Rcpp.h>
#include <numeric>      // for std::partial_sum used below
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

See 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());

    // use STL algorithm
    std::partial_sum(x.begin(), x.end(), res.begin());

    return res;
}
```

Cumulative Sum

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

Or just *Rcpp sugar* for one-line solution:

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

Of course, all results are the same.

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

## [1] TRUE
```

Sugar head and tail

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

The *n* largest values of a vector:

```
#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);
}
```

Sugar head and tail

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

The *n* 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);
}
```

Sugar head and tail

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

Example:

```
sourceCpp("code/sugar_head_tail.cpp")
set.seed(42)
x <- rnorm(5)
x

## [1]  1.3710 -0.5647  0.3631  0.6329  0.4043

top_n(x, 3)

## [1] 0.4043 0.6329 1.3710

bottom_n(x, 3)

## [1] -0.5647  0.3631  0.4043
```

Using the R Rmath functions

See http://gallery.rcpp.org/articles/using-rmath-functions/

Besides all the vectorised d/p/q/r distribution functions available via *Rcpp sugar*, we can also access the R Rmath interface.

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

// [[Rcpp::export]]
NumericVector mypnorm(NumericVector x) {
   int n = x.size();
   NumericVector y(n);

   for (int i=0; i<n; i++)
      y[i] = R::pnorm(x[i], 0.0, 1.0, 1, 0);

   return y;
}
```

Using the R Rmath functions

See `http://gallery.rcpp.org/articles/using-rmath-functions/`

A simple illustration:

```
sourceCpp("code/using-rmath-rng.cpp")
x <- seq(-2, 2)
mypnorm(x)

## [1] 0.02275 0.15866 0.50000 0.84134 0.97725

pnorm(x)

## [1] 0.02275 0.15866 0.50000 0.84134 0.97725
```

The vectorised variants available via *Rcpp sugar* are even more convenient.

Armadillo subsetting

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

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

// [[Rcpp::export]]
mat matrixSubset(mat M) {

    // logical condition: where is transpose larger?
    umat a = trans(M) > M;
    mat N = conv_to<mat>::from(a);

    return N;
}
```

Armadillo subsetting

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

```
M <- matrix(1:9, 3, 3)
M

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

matrixSubset(M)

##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    1    0    0
## [3,]    1    1    0
```

Armadillo subsetting

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

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::export]]
arma::vec matrixSubset2(arma::mat M) {
    arma::mat Z = M * M.t();

    // extracting element matching a condtion
    arma::vec v = Z.elem( arma::find(Z >= 100));
    return v;
}
```

```
matrixSubset2(M)
```

```
##       [,1]
## [1,]   108
## [2,]   108
## [3,]   126
```

A C++11 example: `auto`

See `http://gallery.rcpp.org/articles/first-steps-with-C++11/`

```
#include <Rcpp.h>

// Enable C++11 via plugin (Rcpp >= 0.10.3)
// [[Rcpp::plugins(cpp11)]]

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

A C++11 example: Initialiser lists

See `http://gallery.rcpp.org/articles/first-steps-with-C++11/`

```
#include <Rcpp.h>

// Enable C++11 via plugin (Rcpp >= 0.10.3)
// [[Rcpp::plugins(cpp11)]]

using namespace std::vector;
using namespace std::string;

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

A C++11 example: Loops over ranges

See http://gallery.rcpp.org/articles/first-steps-with-C++11/

```
#include <Rcpp.h>

// Enable C++11 via plugin (Rcpp >= 0.10.3)
// [[Rcpp::plugins(cpp11)]]

// [[Rcpp::export]]
int simpleProd(std::vector<int> vec) {
  int prod = 1;
  for (int &x : vec) {  // loop over all vals of vec
    prod *= x;          // compute product
  }
  return prod;
}
```

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;
}
```

A simple C++ Lambda example

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

An R example use of the 'lambda' function:

```
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/

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

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

using namespace boost::gregorian;

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

Using Boost via BH

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

We can test this from R:

```
sourceCpp("code/boost-bh.cpp")
getIMMDate(9, 2013)

## [1] "2013-09-18"

getIMMDate(3, 2020)

## [1] "2020-03-18"

getIMMDate(12, 2030)

## [1] "2030-12-18"
```

Using Exceptions

See http://gallery.rcpp.org/articles/intro-to-exceptions/

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
double takeLog(double val) {
    try {
        if (val <= 0.0) {                  // log() not defined here
            throw std::range_error("Inadmissible value");
        }
        return log(val);
    } catch(std::exception &ex) {
        forward_exception_to_r(ex);
    } catch(...) {
        ::Rf_error("c++ exception (unknown reason)");
    }
    return NA_REAL;             // not reached
}
```

Using Exceptions

See `http://gallery.rcpp.org/articles/intro-to-exceptions/`

```
takeLog(exp(1))    # works

## [1] 1

takeLog(-1.0)      # throws exception

## Error:  Inadmissible value

takeLog(exp(2))    # but carries on

## [1] 2
```

Calling an R function from C++

See 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;
}
```

Calling an R function from C++

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

R calling a C++ function supplying an R function to be called:

```
sourceCpp("code/r-from-cpp.cpp")
set.seed(42)
x <- rnorm(1e5)
callFunction(x, fivenum)

## [1] -4.043276 -0.682384 -0.002066  0.673325  4.328091

# result of course same as:
fivenum(x)

## [1] -4.043276 -0.682384 -0.002066  0.673325  4.328091
```

Armadillo Eigenvalues

See 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

See 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,]  0.3319
## [2,]  1.6856
## [3,]  2.4099
## [4,] 14.2100


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

Multivariate Normal RNG Draw

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

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

using namespace Rcpp;

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

Passing any R object with ease: Sparse Matrix

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

Define a sparse matrix as an S4 object:

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

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

Passing any R object with ease: Sparse Matrix

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

Access it in C++ with ease:

```
#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);
    // ... code now using Armadillo's sparse matrix
```

Outline

The first example

examples/standard/rinside_sample0.cpp

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

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

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

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

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

    exit(0);
}
```

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.

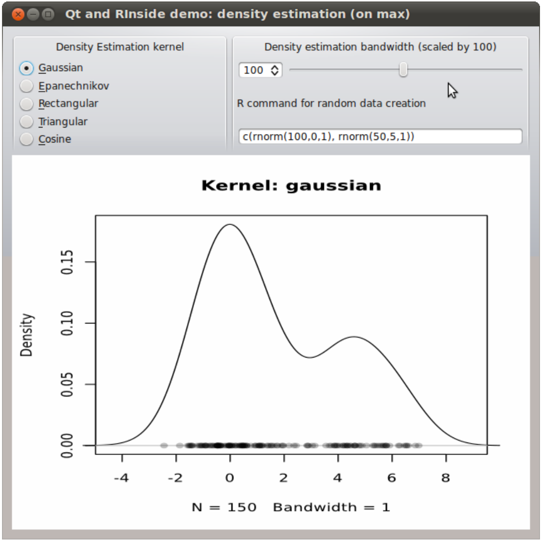
We have a nice example using **Qt**:

```
#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/`



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

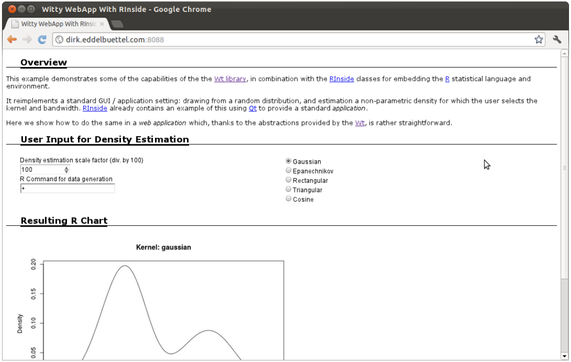
- radio buttons
- sliders
- textentry

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.



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

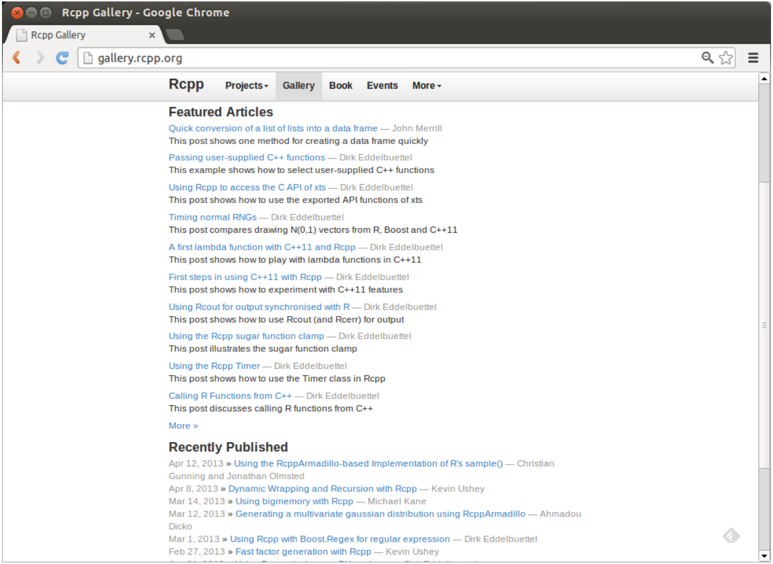
Wt takes care of all browser / app interactions and determines the most featureful deployment.

Outline

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

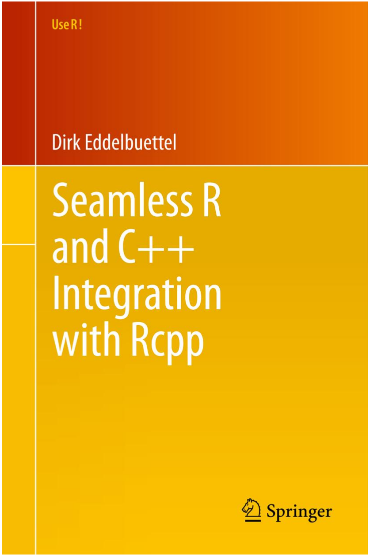
Calling R Functions from C++ — Dirk Eddelbuettel
This post discusses calling R functions from C++

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



In print since June 2013