

HIGHER-PERFORMANCE R PROGRAMMING WITH C++ EXTENSIONS

PART 6: RINSIDE

Dirk Eddebuettel

June 28 and 29, 2017

University of Zürich & ETH Zürich

OVERVIEW

The simplest example:

```
#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 string, ignoring any returns

    exit(0);
}
```

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 used to easily get data in and out from the enclosing C++ program.

- ex2 loads an Rmetrics library and access data
- ex3 run regressions in R, uses coefs and names in C++
- ex4 runs a small portfolio optimisation under risk budgets
- ex5 creates an environment and tests for it
- ex6 illustrations direct data access in R
- ex7 shows `as<>()` conversions from `parseEval()`
- ex8 is another simple bi-directional data access example
- ex9 makes a C++ function accessible to the embedded R
- ex10 creates and alters lists between R and C++
- ex11 uses `RInside` to plot via `curve()`
- ex12 uses `sample()` from C++

PLOTTING EXAMPLE: rinside_sample11.cpp

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

int main(int argc, char *argv[]) {
    RInside R(argc, argv); // create an embedded R instance

    // evaluate an R expression with curve()
    std::string cmd = "tmpf <- tempfile('curve'); png(tmpf); "
        "curve(x^2, -10, 10, 200); "
        "dev.off(); tmpf";
    // by running parseEval, we get the last assignment back, here the filename
    std::string tmpfile = R.parseEval(cmd);

    std::cout << "Could now use plot in " << tmpfile << std::endl;
    unlink(tmpfile.c_str()); // cleaning up

    // alternatively, by forcing a display we can plot to screen
    cmd = "x11(); curve(x^2, -10, 10, 200); Sys.sleep(30);";
    R.parseEvalQ(cmd);
    exit(0);
}
```

PLOTTING EXAMPLE: rinside_sample15.cpp

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

int main(int argc, char *argv[]) {
    RInside R(argc, argv); // create an embedded R instance

    // evaluate an R expression with curve()
    std::string cmd = "library(lattice); "
        "tmpf <- tempfile('xyplot', fileext='.png'); "
        "png(tmpf); "
        "print(xyplot(Girth ~ Height | equal.count(Volume), data=trees)); "
        "dev.off();"
        "tmpf";

    // by running parseEval, we get the last assignment back, here the filename
    std::string tmpfile = R.parseEval(cmd);
    std::cout << "Can now use plot in " << tmpfile << std::endl;

    exit(0);
}
```

MPI

R is famously single-threaded.

High-performance Computing with R frequently resorts to fine-grained (multicore/parallel, doSMP) or coarse-grained (Rmpi, pvm, ...) parallelism. R spawns and controls other jobs.

Jianping Hua suggested to embed R via RInside in MPI applications.

Now we can use the standard and well understood MPI paradigm to launch multiple R instances, each of which is independent of the others.

PARALLEL COMPUTING VIA MPI

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

int main(int argc, char *argv[]) {
    MPI::Init(argc, argv);           // mpi initialization
    int myrank = MPI::COMM_WORLD.Get_rank(); // obtain current node rank
    int nodesize = MPI::COMM_WORLD.Get_size(); // obtain total nodes running.

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

    std::stringstream txt;
    txt << "Hello from node " << myrank // node information
    << " of " << nodesize << " nodes!" << std::endl;

    R["txt"] = txt.str();           // assign string var to R var
    R.parseEvalQ("cat(txt)");       // eval init string

    MPI::Finalize();               // mpi finalization
    exit(0);
}
```

```
$ orterun -n 4 ./rinside_mpi_sample2
Hello from node 0 of 4 nodes!
Hello from node 3 of 4 nodes!
Hello from node 2 of 4 nodes!
Hello from node 1 of 4 nodes!
$
```

This uses Open MPI just locally, other hosts can be added via `-H node1,node2,node3`.

The other example(s) shows how to gather simulation results from MPI nodes.

APPLICATION EXAMPLE: QT

“How to embed R within a larger application” ?

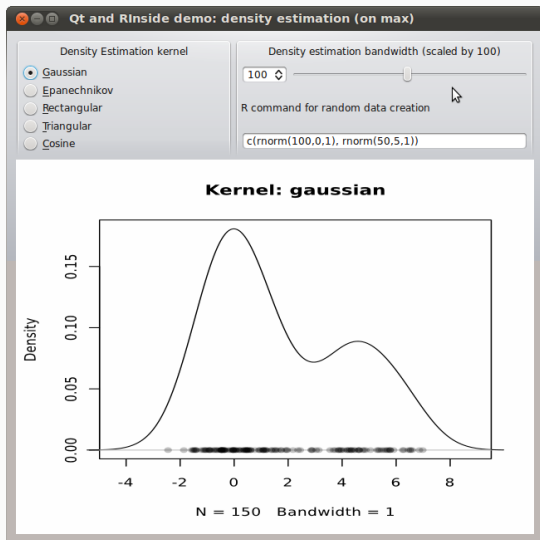
We have an example for Qt.

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

int main(int argc, char *argv[]) {
    RInside R(argc, argv);           // create an emb. R inst

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

APPLICATION EXAMPLE: QT



This uses standard Qt / GUI paradigms of

- radio buttons
- sliders
- textentry

all of which send values to the R process which provides an SVG (or PNG as fallback) image that is plotted.

The actual code is pretty standard Qt / GUI programming (and too verbose to be shown here in full).

The `qtdensity.pro` file is interesting as it maps the entries in the `Makefile` to the Qt standards.

Building is standard `qmake; make` sequence for Qt applications.

APPLICATION EXAMPLE: Wt

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

Density Estimation

Density estimation scale factor (div. by 100)
100

R Command for data generation
`c(rnorm(100,0,1), rnorm(50,5,1))`

Gaussian
 Epanechnikov
 Rectangular
 Triangular
 Cosine

Resulting chart

Kernel: gaussian

Density

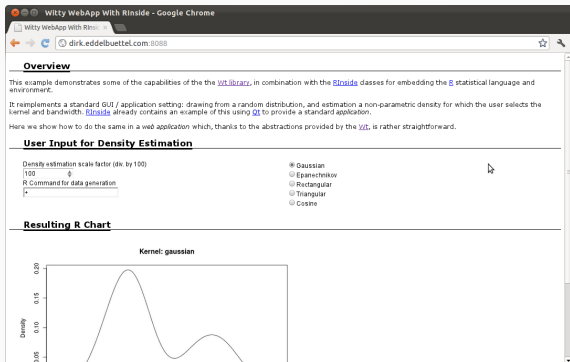
N = 150 Bandwidth = 1

Status

Finished request from 192.168.1.249 using Mozilla/5.0 (Ubuntu; X11; Linux i686; rv:8.0) Gecko/20100101 Firefox/8.0

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.

APPLICATION EXAMPLE: Wt



The screenshot shows a web browser window titled "Witty WebApp With Rinside - Google Chrome" at the URL "dirk.eddelbuettel.com:8088". The page content is as follows:

Overview

This example demonstrates some of the capabilities of the [Wt library](#), in combination with the [Rinside](#) classes for embedding the [R](#) statistical language and environment.

It implements 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

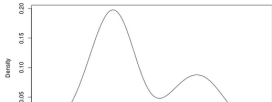
Density estimation scale factor (div. by 100)
100

R Command for data generation

Gaussian
 Epanechnikov
 Rectangular
 Triangular
 Cosine

Resulting R Chart

Kernel: gaussian



Wt can also be “dressed up” with simple CSS styling (and the text displayed comes from an external XML file, further separating content and presentation).

RInside needs headers and libraries from several projects as it

- *embeds R itself* so we need R headers and libraries
- *uses Rcpp* so we need Rcpp headers and libraries
- *used RInside itself* so we also need RInside headers and libraries

The `GNUmakefile` is set-up to create a binary for each example example file supplied. It uses

- `R CMD config` to query all of `--cppflags`, `--ldflags`, `BLAS_LIBS` and `LAPACK_LIBS`
- `Rscript` to query `Rcpp:::CxxFlags` and `Rcpp:::LdFlags`
- `Rscript` to query `RInside:::CxxFlags` and `RInside:::LdFlags`

The `qtdensity.pro` file does the equivalent for Qt.

BUILDING WITH RINSIDE

```
## comment this out if you need a different version of R,
## and set set R_HOME accordingly as an environment variable
R_HOME := $(shell R RHOME)
sources := $(wildcard *.cpp)
programs := $(sources:.cpp=)

## include headers and libraries for R
RCPPLIBS := $(shell $(R_HOME)/bin/R CMD config --cppflags)
RLDFLAGS := $(shell $(R_HOME)/bin/R CMD config --ldflags)
RBLAS := $(shell $(R_HOME)/bin/R CMD config BLAS_LIBS)
RLAPACK := $(shell $(R_HOME)/bin/R CMD config LAPACK_LIBS)

## include headers and libraries for Rcpp interface classes
## note that RCPPLIBS will be empty with Rcpp (>= 0.11.0) and can be omitted
RCPPINCL := $(shell echo 'Rcpp::CxxFlags()' | $(R_HOME)/bin/R --vanilla --slave)
RCPPLIBS := $(shell echo 'Rcpp::LdFlags()' | $(R_HOME)/bin/R --vanilla --slave)

## include headers and libraries for RInside embedding classes
RINSIDEINCL := $(shell echo 'RInside::CxxFlags()' | $(R_HOME)/bin/R --vanilla --slave)
RINSIDELIBS := $(shell echo 'RInside::LdFlags()' | $(R_HOME)/bin/R --vanilla --slave)

## compiler etc settings used in default make rules
CXX := $(shell $(R_HOME)/bin/R CMD config CXX)
CPPFLAGS := -Wall $(shell $(R_HOME)/bin/R CMD config CPPFLAGS)
CXXFLAGS := $(RCPPLIBS) $(RCPPINCL) $(RINSIDEINCL) $(shell $(R_HOME)/bin/R CMD config CXXFLAGS)
LDLIBS := $(RLDFLAGS) $(RRPATH) $(RBLAS) $(RLAPACK) $(RCPPLIBS) $(RINSIDELIBS)
```

SUMMARY

We discussed

- Introduction / Motivation / C++ Basics
- Basic Rcpp Types
- Some key Rcpp ‘Applications’
- A number of Examples from the Gallery
- How to set up packages with Rcpp
- A Brief Discussion of RInside

Rcpp Gallery - Google Chrome

Rcpp Gallery x

gallery.rcpp.org

Rcpp Projects Gallery Book Events More -

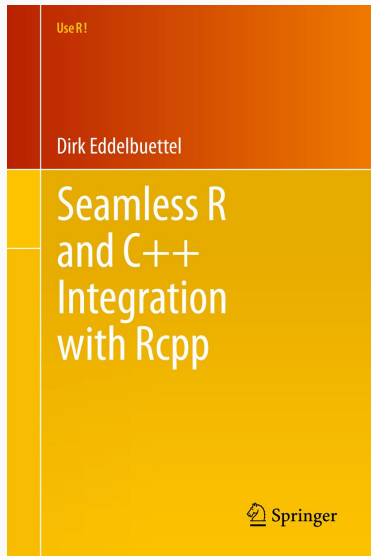
Featured Articles

- [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



On sale since June 2013.

<http://dirk.eddelbuettel.com>

dirk@eddelbuettel.com

[@eddelbuettel](#)