Higher-Performance R via C++

Part 6: RInside

Dirk Eddelbuettel

UZH/ETH Zürich R Courses

June 24-25, 2015

Overview

Overview: Standard

First Example: `rinside_sample0.cpp`

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);
}
```

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

Examples

- 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);
}
```

Overview: MPI

Parallel Computing via 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 indepedent 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);
}
```

Parallel Computing via MPI

```
$ 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" ?

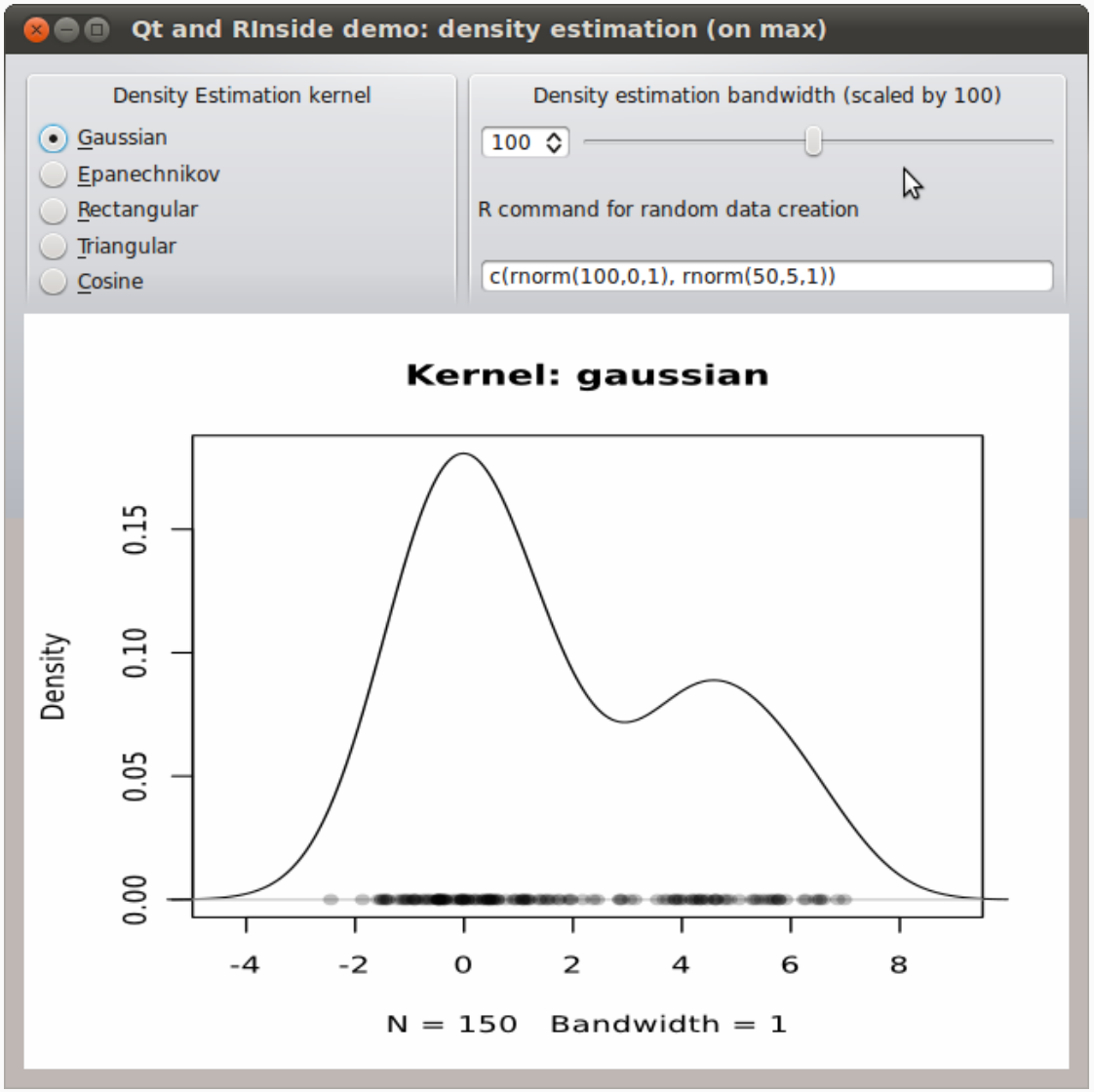
We have an example for Qt.

```
#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.

Application Example: Qt

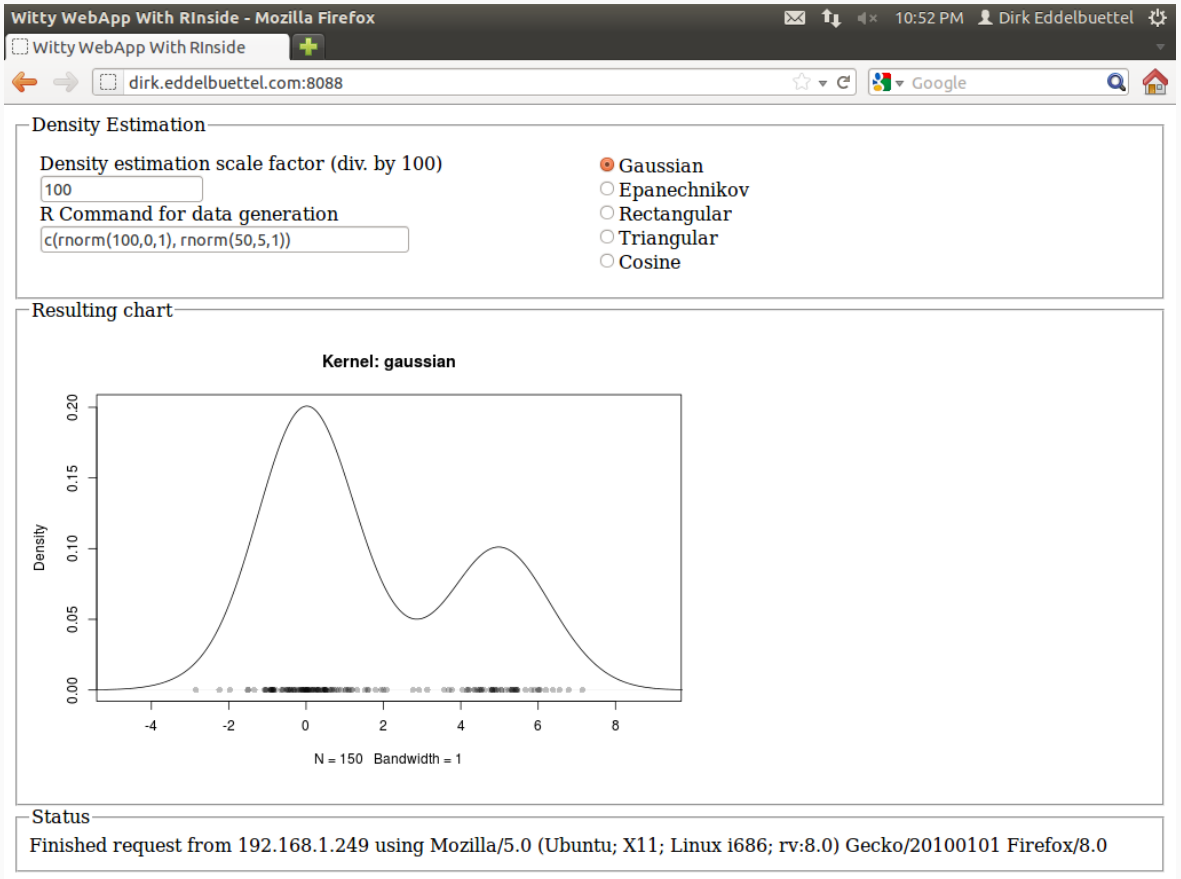
The actual code is pretty standard Qt / GUI programming (and too verbose to be shown here in full).

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

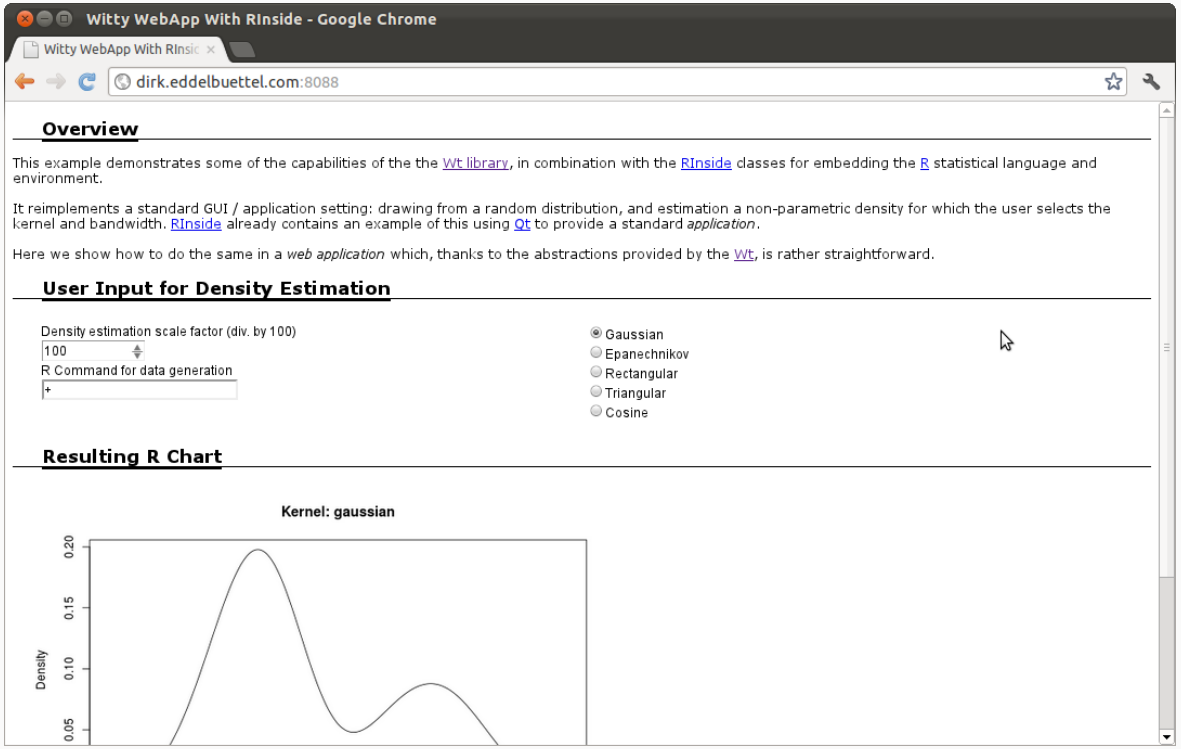
Given the desktop application with Qt, question arises how to deliver something similar *over the web* — and Wt helps.



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



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).

Building with RInside

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

Building with RInside

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
RCPPFLAGS :=    $(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 :=    $(RCPPFLAGS) $(RCPPINCL) $(RINSIDEINCL) $(shell $(R_HOME)/bin/R CMD config CXXFLAGS)
LDLIBS :=       $(RLDFLAGS) $(RRPATH) $(RBLAS) $(RLAPACK) $(RCPPLIBS) $(RINSIDELIBS)

all:            $(programs)
            @test -x /usr/bin/strip && strip $^
```