Rcpp Masterclass / Workshop
Part IV: Applications

Dirk Eddelbuettel[1] Romain François[2]

[1]Debian Project

[2]R Enthusiasts

28 April 2011
preceding *R / Finance 2011*
University of Illinois at Chicago

Outline

The first example

examples/standard/rinside_sample0.cpp

We have seen this first example in part I:

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

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

    RInside R(argc, argv);              // create embedded R inst.

    R["txt"] = "Hello, world!\n";       // assign    to 'txt' in R

    R.parseEvalQ("cat(txt)");           // eval string, ignore result

    exit(0);
}
```

Assign a variable, evaluate an expression—easy!

A second example: part one

examples/standard/rinside_sample1.cpp

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

Rcpp::NumericMatrix createMatrix(const int n) {
    Rcpp::NumericMatrix M(n,n);
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            M(i,j) = i*10 + j;
        }
    }
    return(M);
}
```

A second example: part two

examples/standard/rinside_sample1.cpp

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

  const int mdim = 4;       // let the matrices be 4 by 4; create, fill
  R["M"] = createMatrix(mdim);   // assign data Matrix to R's 'M' var

  std::string str =
    "cat('Running ls()\n'); print(ls()); "
    "cat('Showing M\n'); print(M); "
    "cat('Showing colSums()\n'); Z <- colSums(M); "
    "print(Z); Z";                    // returns Z
  Rcpp::NumericVector v = R.parseEval(str); // eval, assign
  // now show vector on stdout
  exit(0);
}
```

Other example files provide similar R snippets and interchange.

A third example: Calling R plot functions

examples/standard/rinside_sample11.cpp

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

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 filename back
  std::string tmpfile = R.parseEval(cmd);

  std::cout << "Could 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);
}
```

A fourth example: Using Rcpp modules

examples/standard/rinside_module_sample0.cpp

```
#include <RInside.h>                    // for the embedded R via RInside
// a c++ function we wish to expose to R
const char* hello( std::string who ){
    std::string result( "hello " ) ;
    result += who ;
    return result.c_str() ;
}
RCPP_MODULE(bling){
    using namespace Rcpp ;
    function( "hello", &hello );
}
int main(int argc, char *argv[]) {
    // create an embedded R instance -- and load Rcpp so that modules work
    RInside R(argc, argv, true);
    // load the bling module
    R["bling"] = LOAD_RCPP_MODULE(bling) ;
    // call it and display the result
    std::string result = R.parseEval("bling$hello('world')") ;
    std::cout << "bling$hello('world') = '" << result << "'"
              << std::endl ;
    exit(0);
}
```

Other RInside standard examples

A quick overview:

- 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++

Outline

Parallel Computing with RInside

R is famously single-threaded.

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

But somebody's bug may be somebody's else's feature: 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.

A first example

examples/standard/rinside_sample2.cpp

```
#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();        // current node rank
  int nodesize = MPI::COMM_WORLD.Get_size();      // total nodes running.

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

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

  R["txt"] = txt.str();                           // assign to R var 'txt'
  R.parseEvalQ("cat(txt)");                       // eval, ignore returns

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

A first example: Output

examples/standard/rinside_sample2.cpp

```
edd@max:/tmp$ orterun -n 8 ./rinside_mpi_sample2
Hello from node 5 of 8 nodes!
Hello from node 7 of 8 nodes!
Hello from node 1 of 8 nodes!
Hello from node 0 of 8 nodes!
Hello from node 2 of 8 nodes!
Hello from node 3 of 8 nodes!
Hello from node 4 of 8 nodes!
Hello from node 6 of 8 nodes!
edd@max:/tmp$
```

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.

Outline

Application example: Qt

RInside `examples/qt/`

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

We just added a new example using **Qt**:

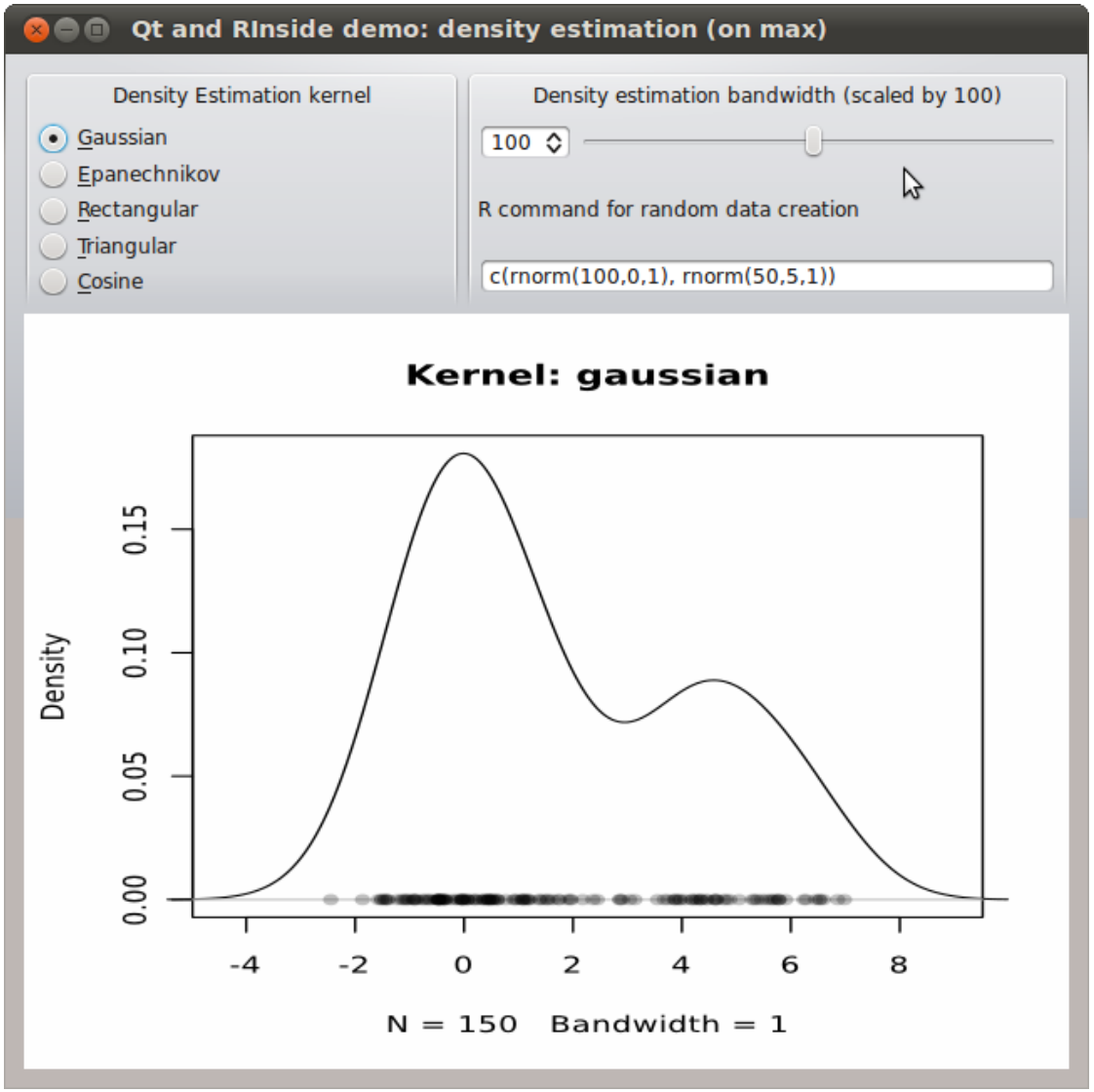
```
#include <QApplication>
#include "qtdensity.h"

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

    QApplication app(argc, argv);
    QtDensity qtdensity(R);   // pass R inst. by reference
    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 an SVG image that is plotted.

Application example: Qt density slider

RInside `examples/qt/`

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

The `qtdensity.pro` file is interesting as it maps the entries in the `Makefile` (discussed in the next section) to the **Qt** standards.

It may need an update for OS X—we have not tried that yet.

Outline

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

RInside itself so we also need RInside headers and libraries

Building with RInside

Use the Makefile in `examples/standard`

The `Makefile` is set-up to create an binary for 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**.

Outline

Armadillo

From `arma.sf.net` and slightly edited

Armadillo is a C++ linear algebra library aiming towards a good balance between speed and ease of use. Integer, floating point and complex numbers are supported, as well as a subset of trigonometric and statistics functions. Various matrix decompositions are provided.

A delayed evaluation approach is employed (during compile time) to combine several operations into one and reduce (or eliminate) the need for temporaries. This is accomplished through recursive templates and template meta-programming.

This library is useful if C++ has been decided as the language of choice (due to speed and/or integration capabilities).

Armadillo highlights

- Provide 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`, and a recent technical report (Sanderson, 2010).
- Modern code, building upon and extending from earlier matrix libraries.
- Responsive and active maintainer, frequent updates.

RcppArmadillo highlights

- Template-only builds—no linking, and available everywhere R and a compiler work (but **Rcpp** is needed to)!
- Easy to use, just add `LinkingTo: RcppArmadillo, Rcpp` to DESCRIPTION (*i.e.*, no added cost beyond **Rcpp**)
- Really easy from R via **Rcpp**
- Frequently updated, easy to use

Outline

Complete file for fastLM

RcppArmadillo `src/fastLm.cpp`

```
#include <RcppArmadillo.h>

extern "C" SEXP fastLm(SEXP ys, SEXP Xs) {
  try {
    arma::colvec y = Rcpp::as<arma::colvec>(ys);  // direct to arma
    arma::mat X    = Rcpp::as<arma::mat>(Xs);
    int df = X.n_rows - X.n_cols;
    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)/df;  // std.errors of coefs
    arma::colvec std_err = arma::sqrt(s2 *
                arma::diagvec(arma::pinv(arma::trans(X)*X)));
    return Rcpp::List::create(Rcpp::Named("coefsficients")=coef,
                              Rcpp::Named("stderr")  = std_err,
                              Rcpp::Named("df")      = df);
  } catch( std::exception &ex ) {
      forward_exception_to_r( ex );
  } catch(...) {
      ::Rf_error( "c++ exception (unknown reason)" );
  }
  return R_NilValue; // -Wall
}
```

Core part of fastLM

RcppArmadillo `src/fastLm.cpp`

```
arma::colvec y = Rcpp::as<arma::colvec>(ys); // to arma
arma::mat X    = Rcpp::as<arma::mat>(Xs);

int df = X.n_rows - X.n_cols;

arma::colvec coef = arma::solve(X, y);          // fit y ~ X

arma::colvec res  = y - X*coef;                 // residuals

double s2 = std::inner_product(res.begin(), res.end(),
                               res.begin(), 0.0)/df;      // std.err coefs

arma::colvec std_err = arma::sqrt(s2 *
                      arma::diagvec(arma::pinv(arma::trans(X)*X)));

return Rcpp::List::create(Rcpp::Named("df") = df,
                          Rcpp::Named("stderr") = std_err,
                          Rcpp::Named("coefficients")     = coef);
```

Easy transfer from (and to) R

RcppArmadillo `src/fastLm.cpp`

```
arma::colvec y = Rcpp::as<arma::colvec>(ys);  // to arma
arma::mat X    = Rcpp::as<arma::mat>(Xs);

int df = X.n_rows - X.n_cols;

arma::colvec coef = arma::solve(X, y);        // fit y ~ X

arma::colvec res  = y - X*coef;               // residuals

double s2 = std::inner_product(res.begin(), res.end(),
                               res.begin(), 0.0)/df;      // std.err coefs

arma::colvec std_err = arma::sqrt(s2 *
                      arma::diagvec(arma::pinv(arma::trans(X)*X)));

return Rcpp::List::create(Rcpp::Named("df") = df,
                          Rcpp::Named("stderr") = std_err,
                          Rcpp::Named("coefficients")     = coef);
```

Easy linear algebra via Armadillo

```
arma::colvec y = Rcpp::as<arma::colvec>(ys);  // to arma
arma::mat X    = Rcpp::as<arma::mat>(Xs);

int df = X.n_rows - X.n_cols;

arma::colvec coef = arma::solve(X, y);        // fit y ~ X

arma::colvec res  = y - X*coef;               // residuals

double s2 = std::inner_product(res.begin(), res.end(),
                               res.begin(), 0.0)/df;      // std.err coefs

arma::colvec std_err = arma::sqrt(s2 *
                       arma::diagvec(arma::pinv(arma::trans(X)*X)));

return Rcpp::List::create(Rcpp::Named("df") = df,
                          Rcpp::Named("stderr") = std_err,
                          Rcpp::Named("coefficients")     = coef);
```

Outline

Example: VAR(1) Simulation

examples/part4/varSimulation.r

Lance Bachmeier started this example for his graduate students: Simulate a VAR(1) model row by row:

```
R> ## parameter and error terms used throughout
R> a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
R> e <- matrix(rnorm(10000),ncol=2)
R> ## Let's start with the R version
R> rSim <- function(coeff, errors) {
+    simdata <- matrix(0, nrow(errors), ncol(errors))
+    for (row in 2:nrow(errors)) {
+      simdata[row,] = coeff %*% simdata[(row-1),] + errors[row,]
+    }
+    return(simdata)
+ }
R> rData <- rSim(a, e)                                  # generated by R
```

Example: VAR(1) Simulation – Compiled R

examples/part4/varSimulation.r

With R 2.13.0, we can also compile the R function:

```
R> ## Now let's load the R compiler (requires R 2.13 or later)
R> suppressMessages(require(compiler))
R> compRsim <- cmpfun(rSim)
R> compRData <- compRsim(a,e)                    # gen. by R 'compiled'
R> stopifnot(all.equal(rData, compRData))        # checking results
```

Example: VAR(1) Simulation – RcppArmadillo

examples/part4/varSimulation.r

```
R> ## Now load 'inline' to compile C++ code on the fly
R> suppressMessages(require(inline))
R> code <- '
+    arma::mat coeff = Rcpp::as<arma::mat>(a);
+    arma::mat errors = Rcpp::as<arma::mat>(e);
+    int m = errors.n_rows; int n = errors.n_cols;
+    arma::mat simdata(m,n);
+    simdata.row(0) = arma::zeros<arma::mat>(1,n);
+    for (int row=1; row<m; row++) {
+      simdata.row(row) = simdata.row(row-1) *
+                         trans(coeff)+errors.row(row);
+    }
+    return Rcpp::wrap(simdata);
+ '
R> ## create the compiled function
R> rcppSim <- cxxfunction(signature(a="numeric",e="numeric"),
+                         code,plugin="RcppArmadillo")
R> rcppData <- rcppSim(a,e)                    # generated by C++ code
R> stopifnot(all.equal(rData, rcppData))       # checking results
```

Example: VAR(1) Simulation – RcppArmadillo

examples/part4/varSimulation.r

```
R> ## now load the rbenchmark package and compare all three
R> suppressMessages(library(rbenchmark))
R> res <- benchmark(rcppSim(a,e),
+                   rSim(a,e),
+                   compRsim(a,e),
+                   columns=c("test", "replications",
+                             "elapsed", "relative"),
+                   order="relative")
R> print(res)
            test replications elapsed relative
1  rcppSim(a, e)          100   0.038   1.0000
3 compRsim(a, e)          100   2.011  52.9211
2     rSim(a, e)          100   4.148 109.1579
R>
```

Example: VAR(1) Simulation – RcppArmadillo

examples/part4/varSimulation.r

```
R> ## now load the rbenchmark package and compare all three
R> suppressMessages(library(rbenchmark))
R> res <- benchmark(rcppSim(a,e),
+                   rSim(a,e),
+                   compRsim(a,e),
+                   columns=c("test", "replications",
+                             "elapsed", "relative"),
+                   order="relative")
R> print(res)
            test replications elapsed relative
1  rcppSim(a, e)          100   0.038   1.0000
3 compRsim(a, e)          100   2.011  52.9211
2     rSim(a, e)          100   4.148 109.1579
R>
```

This is now in SVN for **RcppArmadillo** and will be in the next version.

Outline

RcppGSL

RcppGSL is *almost* as user-friendly as **RcppArmadillo**

But given that the **GSL** is a C library, we need to

- do memory management and free objects
- arrange for the GSL linker to be found

RcppGSL may still be a convenient tool for programmers more familiar with C than C++ wanting to deploy GSL algorithms.

Outline

Vector norm example—*c.f.* GSL manual

examples/part4/gslNorm.cpp

```
#include <RcppGSL.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>

extern "C" SEXP colNorm(SEXP sM) {
  try {
    RcppGSL::matrix<double> M = sM;        // SEXP to gsl data structure
    int k = M.ncol();
    Rcpp::NumericVector n(k);              // to store results
    for (int j = 0; j < k; j++) {
      RcppGSL::vector_view<double> colview =
                          gsl_matrix_column (M, j);
      n[j] = gsl_blas_dnrm2(colview);
    }
    M.free();
    return n;                              // return vector
  } catch( std::exception &ex ) {
    forward_exception_to_r( ex );
  } catch(...) {
    ::Rf_error( "c++ exception (unknown reason)" );
  }
  return R_NilValue; // -Wall
}
```

Core part of example

examples/part4/gslNorm.cpp

```
RcppGSL::matrix<double> M = sM;          // SEXP to GSL data
int k = M.ncol();
Rcpp::NumericVector n(k);                // to store results

for (int j = 0; j < k; j++) {
  RcppGSL::vector_view<double> colview =
                              gsl_matrix_column (M, j);
  n[j] = gsl_blas_dnrm2(colview);
}
M.free();
return n;                                // return vector
```

Core part of example

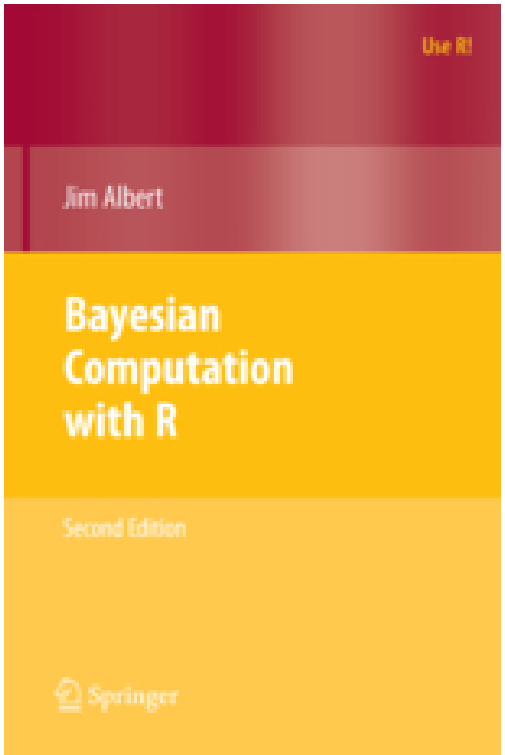
Using standard GSL functions: `examples/part4/gslNorm.cpp`

```
RcppGSL::matrix<double> M = sM;          // SEXP to GSL data
int k = M.ncol();
Rcpp::NumericVector n(k);                // to store results

for (int j = 0; j < k; j++) {
  RcppGSL::vector_view<double> colview =
                              gsl_matrix_column (M, j);
  n[j] = gsl_blas_dnrm2(colview);
}
M.free();
return n;                                // return vector
```

Outline

Accelerating Monte Carlo



Albert. *Bayesian Computation with R*, 2nd ed. Springer, 2009

Albert introduces simulations with a simple example in the first chapter.

We will study this example and translate it to R using RcppArmadillo (and Rcpp).

The idea is to, for a given level α, and sizes *n* and *m*, draw a number *N* of samples at these sizes, compoute a *t*-statistic and record if the test statistic exceeds the theoretical critical value given the parameters.

This allows us to study the impact of varying α, *N* or *M* — as well as varying parameters or even families of the random vectors.

Restating the problem

- With two samples $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ we can test

$$H_0 : \mu_x = \mu_y$$

- With sample means $\bar{X}$ and $\bar{Y}$, and s_x and $_y$ as respective standard deviations, the standard test is

$$T = \frac{\bar{X} - \bar{Y}}{s_P \sqrt{1/m + 1/n}}$$

whew s_p is the pooled standard deviation

$$s_p = \sqrt{\frac{(m-1)s_x^2 + (n-1)s_y^2}{m+n-2}}$$

Restating the problem

- Under H_0, we have $T \sim t(m+n-2)$ provided that
  - x_i and $x+i$ are NID
  - the standard deviations of populations x and y are equal.
- For a given level α, we can reject H if

$$|T| \geq t_{n+m-2,\alpha/2}$$

- But happens when we have
  - unequal population variances, or
  - non-normal distributions?
- Simulations can tell us.

Outline

Basic R version

Core function: `examples/part4/montecarlo.r`

```
## Section 1.3.3
## simulation algorithm for normal populations
sim1_3_3_R <- function() {
    alpha <- .1; m <- 10; n <- 10  # sets alpha, m, n
    N <- 10000                     # sets nb of sims
    n.reject <- 0                  # number of rejections
    crit <- qt(1-alpha/2,n+m-2)
    for (i in 1:N) {
        x <- rnorm(m,mean=0,sd=1)  # simulates xs from population 1
        y <- rnorm(n,mean=0,sd=1)  # simulates ys from population 2
        t.stat <- tstatistic(x,y)  # computes the t statistic
        if (abs(t.stat)>crit)
            n.reject=n.reject+1    # reject if |t| exceeds critical pt
    }
    true.sig.level <- n.reject/N   # est. is proportion of rejections
}
```

Basic R version

Helper function for *t*-statistic: `examples/part4/montecarlo.r`

```
## helper function
tstatistic <- function(x,y) {
    m <- length(x)
    n <- length(y)
    sp <- sqrt(((m-1)*sd(x)^2 + (n-1)*sd(y)^2) / (m+n-2))
    t.stat <- (mean(x) - mean(y)) / (sp*sqrt(1/m + 1/n))
    return(t.stat)
}
```

Outline

RcppArmadillo version

Main function: `examples/part4/montecarlo.r`

```
sim1_3_3_arma <- cxxfunction(, plugin="RcppArmadillo",
                                inc=tstat_arma, body='
  RNGScope scope;      // properly deal with RNGs
  double alpha = 0.1;
  int m = 10, n = 10; // sets alpha, m, n
  int N = 10000;       // sets the number of sims
  double n_reject = 0; // counter of num. of rejects
  double crit = ::Rf_qt(1.0-alpha/2.0, n+m-2.0,true,false);
  for (int i=0; i<N; i++)  {
    NumericVector x = rnorm(m, 0, 1);    // sim xs from pop 1
    NumericVector y = rnorm(n, 0, 1);    // sim ys from pop 2
    double t_stat = tstatistic(Rcpp::as<arma::vec>(x),
                               Rcpp::as<arma::vec>(y));
    if (fabs(t_stat) > crit)
      n_reject++;              // reject if |t| exceeds critical pt
  }
  double true_sig_level = 1.0*n_reject / N; // est. prop rejects
  return(wrap(true_sig_level));
')
```

RcppArmadillo version

Helper function for *t*-statistic: : `examples/part4/montecarlo.r`

```
tstat_arma <- '
  double tstatistic(const arma::vec &x, const arma::vec &y) {
    int m = x.n_elem;
    int n = y.n_elem;
    double sp = sqrt( ( (m-1.0)*pow(stddev(x),2) +
                        (n-1)*pow(stddev(y),2) ) / (m+n-2.0) );
    double t_stat = (mean(x)-mean(y))/(sp*sqrt(1.0/m+1.0/n));
    return(t_stat);
  }
'
```

Rcpp version—using SVN version with `mean`, `sd`, ...

Main function: `examples/part4/montecarlo.r`

```
sim1_3_3_rcpp <- cxxfunction(, plugin="Rcpp",
                                inc=tstat_rcpp, body='
  RNGScope scope;      // properly deal with RNG settings
  double alpha = 0.1;
  int m = 10, n = 10; // sets alpha, m, n
  int N = 10000;       // sets the number of simulations
  double n_reject = 0; // counter of num. of rejections
  double crit = ::Rf_qt(1.0-alpha/2.0, n+m-2.0, true, false);
  for (int i=0; i<N; i++)  {
    NumericVector x = rnorm(m, 0, 1);    // sim xs from pop 1
    NumericVector y = rnorm(n, 0, 1);    // sim ys from pop 2
    double t_stat = tstatistic(x, y);
    if (fabs(t_stat) > crit)
      n_reject++;          // reject if |t| exceeds critical pt
  }
  double true_sig_level = 1.0*n_reject / N; // est. prop rejects
  return(wrap(true_sig_level));
')
```

Outline

Rcpp version—using SVN version with `mean`, `sd`, ...

Helper function: `examples/part4/montecarlo.r`

```
tstat_rcpp <- '
 double tstatistic(const NumericVector &x,
                   const NumericVector &y) {
   int m = x.size();
   int n = y.size();
   double sp = sqrt( ( (m-1.0)*pow(sd(x),2) +
                     (n-1)*pow(sd(y),2) ) / (m+n-2.0) );
   double t_stat = (mean(x)-mean(y))/(sp*sqrt(1.0/m+1.0/n));
   return(t_stat);
 }
'
```

Outline

Benchmark results

examples/part4/montecarlo.r

```
R> library(rbenchmark)
R> res <- benchmark(sim1_3_3_R(),
+                   sim1_3_3_Rcomp(),
+                   sim1_3_3_arma(),
+                   sim1_3_3_rcpp(),
+                   columns=c("test", "replications",
+                             "elapsed", "relative",
+                             "user.self"),
+                   order="relative")
R> res
              test replications elapsed relative user.self
3  sim1_3_3_arma()          100   2.118  1.00000      2.12
4  sim1_3_3_rcpp()          100   2.192  1.03494      2.19
1     sim1_3_3_R()          100 153.772 72.60246    153.70
2 sim1_3_3_Rcomp()          100 154.251 72.82861    154.19
R>
```

Benchmark results

```
R> res
               test replications elapsed relative user.self
3  sim1_3_3_arma()          100   2.118  1.00000      2.12
4  sim1_3_3_rcpp()          100   2.192  1.03494      2.19
1     sim1_3_3_R()          100 153.772 72.60246    153.70
2 sim1_3_3_Rcomp()          100 154.251 72.82861    154.19
R>
```

In this example, the R compiler does not help at all. The difference between **RcppArmadillo** and **Rcpp** is neglible.

Suggestions (by Albert): replace *n*, *m*, standard deviations of Normal RNG, replace Nornal RNG, ... which, thanks to **Rcpp** and 'Rcpp sugar' is a snap.

Simulation results

examples/part4/montecarlo.r

Albert reports this table:

Populations	True Sign. Level
Normal pop. with equal spreads	0.0986
Normal pop. with unequal spreads	0.1127
$t(4)$ distr. with equal spreads	0.0968
Expon. pop. with equal spreads	0.1019
Normal + exp. pop. with unequal spreads	0.1563

Table: True significance level of t-test computed by simulation; standard error of each estimate is approximately 0.003.

Given that our simulations are $\approx$ 70-times faster, we can reduce the standard error to $\sqrt{0.1 \times 0.9/1,000,000} = 0.0003$.

That's it, folks!

Want to learn more ?

- Check the vignettes
- Ask questions on the `Rcpp-devel` mailing list
- Inquire about commercial support

Romain François **romain@r-enthusiasts.com**
Dirk Eddelbuettel **edd@debian.org**