Rcpp Workshop
Part IV: Applications

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

Sponsored by ASA, CTSI and PCOR
Medical College of Wisconsin
Milwaukee, WI
May 11, 2013
Outline

1. RInside
   - Basics
     - MPI
     - Qt
     - Wt
     - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
The first example
examples/standard/rinside_sample0.cpp

We have seen this first example in part I:

```cpp
#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!
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 to easily get data in and out from the enclosing C++ program.
A second example: part one
examples/standard/rinside_sample1.cpp

```c++
#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);
}
```

Dirk Eddelbuettel
Rcpp Workshop — Part IV: Applications
A second example: part two
examples/standard/rinside_sample1.cpp

```c++
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.
#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);
}
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 \texttt{as<>()} conversions from \texttt{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++
- ex12 uses \texttt{sample()} from C++
Outline

1 RInside
   • Basics
   • MPI
   • Qt
   • Wt
   • Building with RInside

2 RcppArmadillo
   • Armadillo
   • Example: FastLM
   • Example: VAR(1) Simulation
   • Example: Kalman Filter
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.

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.
A first example
examples/standard/rinside_sample2.cpp

```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 << " 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);
}
```

Dirk Eddelbuettel  Rcpp Workshop — Part IV: Applications
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

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
The question is sometimes asked how to embed **RInside** in a larger program.

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

```cpp
#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

This uses standard Qt / GUI paradigms of
- radio buttons
- sliders
- text entry
all of which send values to the R process which provides an SVG (or PNG as fallback) image that is plotted.

Dirk Eddelbuettel
Rcpp Workshop — Part IV: Applications
Application example: Qt density slider

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

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
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.
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).
Outline

1. **RInside**
   - Basics
   - MPI
   - Qt
   - Wt
   - **Building with RInside**

2. **RcppArmadillo**
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
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**.
Building with RInside

## comment out if you need a different version of R, and set R_HOME
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)

## if you need to set an rpath to R itself, also uncomment
#RRPATH := -Wl,-rpath,$(R_HOME)/lib

## include headers and libraries for Rcpp interface classes
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)
LDLIBS := $(RLDFLAGS) $(RRPATH) $(RBLAS) $(RLAPACK) $(RCPPLIBS) $(RINSIDELIBS)

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

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
What is Armadillo?

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

- Provides 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 wherever 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

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
#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 $y \sim X$
        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("coefficients")=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
}
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 \sim 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

```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

```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);
```
One note on direct casting with Armadillo

The code as just shown:

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

is very convenient, but does incur an additional copy of each object. A lighter variant uses two steps in which only a pointer to the object is copied:

```cpp
Rcpp::NumericVector yr(ys);
Rcpp::NumericMatrix Xr(Xs);
int n = Xr.nrow(), k = Xr.ncol();
arma::mat X(Xr.begin(), n, k, false);
arma::colvec y(yr.begin(), yr.size(), false);
```

If performance is a concern, the latter approach may be preferable.
Performance comparison

Running the script included in the **RcppArmadillo** package:

```
edd@max:~/svn/rcpp/pkg/RcppArmadillo/inst/examples$ r fastLm.r
Loading required package: methods
   test  replications relative elapsed
   1  fLmOneCast(X, y)        5000 1.000000 0.170
   2  fLmTwoCasts(X, y)       5000 1.029412 0.175
   4  fastLmPureDotCall(X, y) 5000 1.211765 0.206
   3  fastLmPure(X, y)        5000 2.235294 0.380
   6  lm.fit(X, y)            5000 3.911765 0.665
   5  fastLm(frm, data = trees) 5000 40.488235 6.883
   7  lm(frm, data = trees)   5000 53.735294 9.135
edd@max:~/svn/rcpp/pkg/RcppArmadillo/inst/examples$
```

NB: This includes a minor change in SVN and not yet in the released package.
Outline

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
Lance Bachmeier started this example for his graduate students: Simulate a VAR(1) model row by row:

```r
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) {
R+ simdata <- matrix(0, nrow(errors), ncol(errors))
R+ for (row in 2:nrow(errors)) {
R+ simdata[row,] = coeff %*% simdata[(row-1),] + errors[row,]
R+ }
R+ return(simdata)
R+ }
R> rData <- rSim(a, e) # generated by 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)
> suppressMessages(require(compiler))
> compRsim <- cmpfun(rSim)
> compRData <- compRsim(a,e) # gen. by R 'compiled'
> 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
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  1.00000
3 compRsim(a, e)     100  2.011 52.92110
2       rSim(a, e)     100 4.148 109.15792
R>

So more than fifty times faster than byte-compiled R and more than hundred times faster than R code.
## now load the rbenchmark package and compare all three
suppressMessages(library(rbenchmark))
res <- benchmark(rcppSim(a,e),
+   rSim(a,e),
+   compRsrm(a,e),
+   columns=c("test", "replications", "elapsed", "relative"),
+   order="relative")
print(res)

    test  replications elapsed relative
  1 rcppSim(a, e)     100  0.038 1.00000
  3 compRsrm(a, e)    100  2.011 52.9211
  2    rSim(a, e)     100  4.148 109.1579
Outline

1. RInside
   - Basics
   - MPI
   - Qt
   - Wt
   - Building with RInside

2. RcppArmadillo
   - Armadillo
   - Example: FastLM
   - Example: VAR(1) Simulation
   - Example: Kalman Filter
Kalman Filter

The Mathworks has a nice example\(^1\) of a classic 'object tracking' problem showing gains from going from Matlab code to compiled C code.

The example is short:

```matlab
function y = kalmanfilter(z)
  % Predicted state and covariance
  x_prd = A * x_est;
  p_prd = A * p_est * A' + Q;
  % Estimation
  S = H * p_prd' * H' + R;
  B = H * p_prd';
  klm_gain = (S \ B)';
  % Estimated state and covariance
  x_est = x_prd+klm_gain*(z-H*x_prd);
  p_est = p_prd-klm_gain*H*p_prd;
  % Compute the estimated measurements
  y = H * x_est;
end
```

Kalman Filter: In R

Easy enough – first naive solution

FirstKalmanR <- \textbf{function}(pos) \{ 

kf <- \textbf{function}(z) \{ 
  dt <- 1

  A <- \text{matrix}(c(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 6), byrow=TRUE)
  H <- \text{matrix}(c(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 6), byrow=TRUE)
  Q <- \text{diag}(6)
  R <- 1000 * \text{diag}(2)

  N <- \text{nrow}(pos)
  y <- \text{matrix}(\text{NA}, N, 2)

  ## predicted state and covariance
  xprd <- A %*% xest
  pprd <- A %*% pest %*% t(A) + Q

  S <- H %*% t(pprd) %*% t(H) + R
  B <- H %*% t(pprd)

  ## kalmangain <- (S \ B)'
  kg <- t(solve(S, B))

  ## est. state and cov, assign to vars in parent env
  xest <- xest + kg %*% (z-H%*%xprd)
  pest <- pprd - kg %*% H%*%pprd

  ## compute the estimated measurements
  y <- H %*% xest

  for (i in 1:N) {
    y[i,] <- kf(t(pos[i,"drop=FALSE")))
  }

  invisible(y)
\}

\textbf{Dirk Eddelbuettel}
Kalman Filter: In R
Easy enough – with some minor refactoring

KalmanR <- function(pos) {
  kf <- function(z) {
    ## predicted state and covariance
    xprd <- A %*% xest
    pprd <- A %*% pest %*% t(A) + Q

    ## estimation
    S <- H %*% t(pprd) %*% t(H) + R
    B <- H %*% t(pprd)
    ## kg <- (S \ B)'
    kg <- t(solve(S, B))

    ## estimated state and covariance
    ## assigned to vars in parent env
    xest <<- xprd + kg %*% (z-H%*%xprd)
    pest <<- pprd - kg %*% H %*% pprd

    ## compute the estimated measurements
    y <- H %*% xest
  }
  dt <- 1
}

A <- matrix(c(1, 0, dt, 0, 0, 0, # x
              0, 1, 0, dt, 0, 0, # y
              0, 0, 1, 0, dt, 0, # Vx
              0, 0, 0, 1, 0, dt, # Vy
              0, 0, 0, 0, 1, 0, # Ax
              0, 0, 0, 0, 0, 1), # Ay
            6, 6, byrow=TRUE)

H <- matrix(c(1, 0, 0, 0, 0, 0,
              0, 1, 0, 0, 0, 0),
            2, 6, byrow=TRUE)

Q <- diag(6)
R <- 1000 * diag(2)

N <- nrow(pos)
y <- matrix(NA, N, 2)

xest <- matrix(0, 6, 1)
pest <- matrix(0, 6, 6)

for (i in 1:N) {
  y[i,] <- kf(t(pos[i,drop=FALSE]))
}
invisible(y)
Kalman Filter: In C++
Using a simple class

```cpp
using namespace arma;

class Kalman {
private:
    mat A, H, Q, R, xest, pest;
    double dt;

public:
    // constructor, sets up data structures
    Kalman() : dt(1.0) {
        A.eye(6,6);
        A(0,2)=A(1,3)=A(2,4)=A(3,5)=dt;
        H.zeros(2,6);
        H(0,0) = H(1,1) = 1.0;
        Q.eye(6,6);
        R = 1000 * eye(2,2);
        xest.zeros(6,1);
        pest.zeros(6,6);
    }

    // sole member function: estimate model
    mat estimate(const mat & Z) {
        unsigned int n = Z.n_rows,
                    k = Z.n_cols;
        mat Y = zeros(n, k);

        for (unsigned int i = 0; i<n; i++) {
            colvec z = Z.row(i).t();

            // predicted state and covariance
            mat xprd = A * xest;
            mat pprd = A * pest * A.t() + Q;

            // estimation
            mat S = H * pprd.t() * H.t() + R;
            mat B = H * pprd.t();
            mat kg = trans(solve(S, B));

            // estimated state and covariance
            xest = xprd + kg * (z - H * xprd);
            pest = pprd - kg * H * pprd;

            // compute the estimated measurements
            colvec y = H * xest;
            Y.row(i) = y.t();
        }

        return Y;
    }
};
```
Given the code from the previous slide in a text variable `kalmanClass`, we just do this

```r
kalmanSrc <- ' 
mat Z = as<mat>(ZS); // passed from R
Kalman K;
mat Y = K.estimate(Z);
return wrap(Y);
',

KalmanCpp <- cxxfunction(signature(ZS="numeric"),
body=kalmanSrc,
include=kalmanClass,
plugin="RcppArmadillo")
```
Even byte-compiled 'better' R version is 66 times slower:

```r
R> FirstKalmanRC <- cmpfun(FirstKalmanR)
R> KalmanRC <- cmpfun(KalmanR)
R>
R> stopifnot(identical(KalmanR(pos), KalmanRC(pos)),
             all.equal(KalmanR(pos), KalmanCpp(pos)),
             identical(FirstKalmanR(pos), FirstKalmanRC(pos)),
             all.equal(KalmanR(pos), FirstKalmanR(pos)))
R>
R> res <- benchmark(KalmanR(pos), KalmanRC(pos),
                     FirstKalmanR(pos), FirstKalmanRC(pos),
                     KalmanCpp(pos),
                     columns = c("test", "replications", "elapsed", "relative"),
                     order="relative",
                     replications=100)
R>
R> print(res)
```

```r
  test replications elapsed relative
  5 KalmanCpp(pos)        100  0.087  1.0000
  2 KalmanRC(pos)         100  5.774  66.3678
  1 KalmanR(pos)          100  6.448  74.1149
  4 FirstKalmanRC(pos)    100  8.153  93.7126
  3 FirstKalmanR(pos)     100  8.901 102.3103
```
Kalman Filter: Figure

Last but not least we can redo the plot as well.
Outline

3 RcppEigen
   - Overview
   - Example

4 RcppGSL
   - Overview
   - Example
RcppEigen wraps the Eigen library for linear algebra.

Eigen is similar to Armadillo, and very highly optimised—by internal routines replacing even the BLAS for performance.

Eigen is also offering a more complete API than Armadillo (but I prefer to work with the simpler Armadillo, most of the time).

RcppEigen is written mostly by Doug Bates who needs sparse matrix support for his C++ rewrite of lme4 (e.g. lme4eigen).

Eigen can be faster than Armadillo. Andreas Alfons’ CRAN package robustHD (using Armadillo) with a drop-in replacement sparseLTSEigen sees gain of 1/4 to 1/3.

However, Eigen is not always available on all platforms as there can be issues with older compilers (eg on OS X).
Outline

3. RcppEigen
   - Overview
   - Example

4. RcppGSL
   - Overview
   - Example
RcppEigen’s fastLm
Slightly simplified / shortened

```cpp
const MMatrixXd X(as<MMatrixXd>(Xs));
cost MVectorXd y(as<MVectorXd>(ys));
Index n = X.rows(), p = X.cols();
lm ans = do_lm(X, y, ::Rf_asInteger(type));
NumericVector coef = wrap(ans.coef());
List dimnames = NumericMatrix(Xs).attr("dimnames");
VectorXd resid = y - ans.fitted();
double s2 = resid.squaredNorm()/ans.df();
PermutationType Pmat = PermutationType(p);
Pmat.indices() = ans.perm();
VectorXd dd = Pmat * ans.unsc().diagonal();
VectorXd se = (dd.array() * s2).sqrt();
return List::create(_["coefficients"] = coef,
                    _["se"] = se,
                    _["rank"] = ans.rank(),
                    _["df.residual"] = ans.df(),
                    _["perm"] = ans.perm(),
                    _["residuals"] = resid,
                    _["s2"] = s2,
                    _["fitted.values"] = ans.fitted(),
                    _["unsc"] = ans.unsc());
```
Doug defines a base class `lm` from which the following classes derive:

- LLt (standard Cholesky decomposition)
- LDLt (robust Cholesky decomposition with pivoting)
- SymmEigen (standard Eigen-decomposition)
- QR (standard QR decomposition)
- ColPivQR (Householder rank-revealing QR decomposition with column-pivoting)
- SVD (standard SVD decomposition)

The example file `lmBenchmark.R` in the package runs through these.
RcppEigen’s fastLm (cont.)

The benchmark results

<table>
<thead>
<tr>
<th></th>
<th>test</th>
<th>relative</th>
<th>elapsed</th>
<th>user.self</th>
<th>sys.self</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>LDLt</td>
<td>1.000000</td>
<td>0.911</td>
<td>0.91</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>LLt</td>
<td>1.000000</td>
<td>0.911</td>
<td>0.91</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>SymmEig</td>
<td>2.833150</td>
<td>2.581</td>
<td>2.17</td>
<td>0.40</td>
</tr>
<tr>
<td>6</td>
<td>QR</td>
<td>5.050494</td>
<td>4.601</td>
<td>4.17</td>
<td>0.41</td>
</tr>
<tr>
<td>2</td>
<td>ColPivQR</td>
<td>5.102086</td>
<td>4.648</td>
<td>4.20</td>
<td>0.43</td>
</tr>
<tr>
<td>8</td>
<td>arma</td>
<td>6.837541</td>
<td>6.229</td>
<td>6.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>lm.fit</td>
<td>9.189901</td>
<td>8.372</td>
<td>7.12</td>
<td>1.14</td>
</tr>
<tr>
<td>4</td>
<td>SVD</td>
<td>32.183315</td>
<td>29.319</td>
<td>28.44</td>
<td>0.76</td>
</tr>
<tr>
<td>9</td>
<td>GSL</td>
<td>113.680571</td>
<td>103.563</td>
<td>102.42</td>
<td>0.53</td>
</tr>
</tbody>
</table>

This improves significantly over the Armadillo-based solution.
One last remark on the fastLm routines

Doug sometimes reminds us about the occasional fine differences between statistical numerical analysis and standard numerical analysis.

Pivoting schemes are a good example. R uses a custom decomposition (with pivoting) inside of `lm()` which makes it both robust and precise, particularly for rank-deficient matrices.

The example for `fastLm` in both `RcppArmadillo` and `RcppEigen` provides an illustration.

If you are really sure your data is well-behaved, then using a faster (non-pivoting) scheme as in `RcppArmadillo`
Outline

3 RcppEigen
   - Overview
   - Example

4 RcppGSL
   - Overview
   - Example
RcppGSL is a convenience wrapper for accessing the GNU GSL, particularly for vector and matrix functions.

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

3 RcppEigen
   - Overview
   - Example

4 RcppGSL
   - Overview
   - Example
Vector norm example—c.f. GSL manual
examples/part4/gslNorm.cpp

```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
}
```

Dirk Eddelbuettel  
Rcpp Workshop — Part IV: Applications
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
Example: Gibbs Sampler

- Intro
- R
- Rcpp
- RcppGSL
- Performance

Example: Simulations

- Intro
- R
- RcppArmadillo
- Rcpp
- Performance
Darren Wilkinson wrote a couple of blog posts illustrating the performance of different implementations (C, Java, Python, ...) for a simple MCMC Gibbs sampler of this bivariate density:

\[ f(x, y) = kx^2 \exp(-xy^2 - y^2 + 2y - 4x) \]

with conditional distributions

\[ f(x|y) \sim \text{Gamma}(3, y^2 + 4) \]
\[ f(y|x) \sim N\left(\frac{1}{1 + x}, \frac{1}{2(1 + x)}\right) \]

i.e. we need repeated RNG draws from both a Gamma and a Gaussian distribution.
Outline

Example: Gibbs Sampler
- Intro
- R
- Rcpp
- RcppGSL
- Performance

Example: Simulations
- Intro
- R
- RcppArmadillo
- Rcpp
- Performance
Sanjog Misra then sent me working R and C++ versions which I extended. In R we use:

```r
Rgibbs <- function (N, thin) {
  mat <- matrix(0, ncol=2, nrow=N)
  x <- 0
  y <- 0
  for (i in 1:N) {
    for (j in 1:thin) {
      x <- rgamma(1, 3, y*y + 4)
      y <- rnorm(1, 1/(x+1), 1/sqrt(2*(x+1)))
    }
    mat[i,] <- c(x, y)
  }
  mat
}
```
Outline

5 Example: Gibbs Sampler
   • Intro
   • R
   • Rcpp
   • RcppGSL
   • Performance

6 Example: Simulations
   • Intro
   • R
   • RcppArmadillo
   • Rcpp
   • Performance
Gibbs Sampler Example (cont.)

The C++ version using **Rcpp** closely resembles the **R** version:

```c
// n and thin are SEXP's which the Rcpp::as function maps to C++ vars
int N = as<int>(n);
int thn = as<int>(thin);
int i, j;
NumericMatrix mat(N, 2);
RNGScope scope; // Initialize Random number generator
double x=0, y=0;
for (i=0; i<N; i++) {
    for (j=0; j<thn; j++) {
        x = ::Rf_rgamma(3.0, 1.0/(y*y+4));
        y = ::Rf_rnorm(1.0/(x+1), 1.0/sqrt(2*x+2));
    }
    mat(i,0) = x;
    mat(i,1) = y;
}
return mat; // Return to R
```

Dirk Eddelbuettel
Rcpp Workshop — Part IV: Applications
We compile the C++ function:

```r
# Compile and Load
RcppGibbs <- cxxfunction(signature(n="int",
                         thin = "int"),
                         gibbscode, plugin="Rcpp")
```
Outline

5 Example: Gibbs Sampler
   - Intro
   - R
   - Rcpp
   - RcppGSL
   - Performance

6 Example: Simulations
   - Intro
   - R
   - RcppArmadillo
   - Rcpp
   - Performance
We also create a similar variant using the GSL’s random number generators (as in Darren’s example):

gslgibbscode <- '
    int N = as<int>(ns);
    int thin = as<int>(thns);
    int i, j;
    gsl_rng *r = gsl_rng_alloc(gsl_rng_mt19937);
    double x=0, y=0;
    NumericMatrix mat(N, 2);
    for (i=0; i<N; i++) {
        for (j=0; j<thin; j++) {
            x = gsl_ran_gamma(r,3.0,1.0/(y*y+4));
            y = 1.0/(x+1)+gsl_ran_gaussian(r,1.0/sqrt(2*x+2));
        }
        mat(i,0) = x;
        mat(i,1) = y;
    }
    gsl_rng_free(r);
    return mat;       // Return to R
'
We compile the GSL / C function:

```csharp
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

using namespace Rcpp; // just to be explicit
```

```r
## Compile and Load
GSLGibbs <- cxxfunction(signature(ns="int",
                         thns = "int"),
                        body=gslgibbscode,
                        includes=gslgibbsincl,
                        plugin="RcppGSL")
```
5 Example: Gibbs Sampler
- Intro
- R
- Rcpp
- RcppGSL
- Performance

6 Example: Simulations
- Intro
- R
- Rcpp Armadillo
- Rcpp
- Performance
The result show a dramatic gain from the two compiled version relative to the R version, and the byte-compiled R version:

<table>
<thead>
<tr>
<th>test</th>
<th>repl.</th>
<th>elapsed</th>
<th>relative</th>
<th>user.self</th>
<th>sys.self</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 GSLGibbs(N, thn)</td>
<td>10</td>
<td>7.918</td>
<td>1.000000</td>
<td>7.87</td>
<td>0.00</td>
</tr>
<tr>
<td>3 RcppGibbs(N, thn)</td>
<td>10</td>
<td>12.300</td>
<td>1.553423</td>
<td>12.25</td>
<td>0.00</td>
</tr>
<tr>
<td>2 RCgibbs(N, thn)</td>
<td>10</td>
<td>306.349</td>
<td>38.690200</td>
<td>305.07</td>
<td>0.11</td>
</tr>
<tr>
<td>1 Rgibbs(N, thn)</td>
<td>10</td>
<td>412.467</td>
<td>52.092321</td>
<td>410.76</td>
<td>0.18</td>
</tr>
</tbody>
</table>

The gain of the **GSL** version relative to the **Rcpp** is due almost entirely to a much faster RNG for the gamma distribution as shown by `timeRNGs.R`.
Outline

5 Example: Gibbs Sampler
   - Intro
   - R
   - Rcpp
   - RcppGSL
   - Performance

6 Example: Simulations
   - Intro
   - R
   - RcppArmadillo
   - Rcpp
   - Performance
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 $\alpha$, and sizes $n$ and $m$, draw a number $N$ of samples at these sizes, compute 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 $\alpha$, $N$ 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 $s_y$ 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 $\alpha$, 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

5 Example: Gibbs Sampler
   • Intro
   • R
   • Rcpp
   • RcppGSL
   • Performance

6 Example: Simulations
   • Intro
   • R
   • RcppArmadillo
   • Rcpp
   • Performance
## Section 1.3.3
**simulation algorithm for normal populations**

```r
# sets alpha, m, n
# sets nb of sims
# number of rejections
sim1_3_3_R <- function () {
  alpha <- .1; m <- 10; n <- 10
  N <- 10000
  n.reject <- 0
  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) # reject if |t| exceeds critical pt
      n.reject=n.reject+1
  }
  true.sig.level <- n.reject/N # est. is proportion of rejections
}
```
## 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

5 Example: Gibbs Sampler
  - Intro
  - R
  - Rcpp
  - RcppGSL
  - Performance

6 Example: Simulations
  - Intro
  - R
  - RcppArmadillo
  - Rcpp
  - Performance
RcppArmadillo version

Main function: examples/part4/montecarlo.r

sim1_3_3_arma <- cxxfunction(), plugin="RcppArmadillo",
icn=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));
'}
tstat_arma <-

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

5 Example: Gibbs Sampler
   - Intro
   - Rcpp
   - RcppGSL
   - Performance

6 Example: Simulations
   - Intro
   - R
   - RcppArmadillo
   - Rcpp
   - Performance
Rcpp version—using sugar functions `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)  // reject if |t| exceeds critical pt
    n_reject++;
}
double true_sig_level = 1.0*n_reject / N;  // est. prop rejects
return(wrap(true_sig_level));''
```

Dirk Eddelbuettel  
Rcpp Workshop — Part IV: Applications
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

5 Example: Gibbs Sampler
   - Intro
   - R
   - Rcpp
   - RcppGSL
   - Performance

6 Example: Simulations
   - Intro
   - R
   - RcppArmadillo
   - Rcpp
   - Performance
library(rbenchmark)
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")
res

<table>
<thead>
<tr>
<th>test</th>
<th>replications</th>
<th>elapsed</th>
<th>relative</th>
<th>user.self</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim1_3_3_arma()</td>
<td>100</td>
<td>2.118</td>
<td>1.00000</td>
<td>2.12</td>
</tr>
<tr>
<td>sim1_3_3_rcpp()</td>
<td>100</td>
<td>2.192</td>
<td>1.03494</td>
<td>2.19</td>
</tr>
<tr>
<td>sim1_3_3_R()</td>
<td>100</td>
<td>153.772</td>
<td>72.60246</td>
<td>153.70</td>
</tr>
<tr>
<td>sim1_3_3_Rcomp()</td>
<td>100</td>
<td>154.251</td>
<td>72.82861</td>
<td>154.19</td>
</tr>
</tbody>
</table>
### Benchmark results

```r
R> res

<table>
<thead>
<tr>
<th>Test function</th>
<th>Replications</th>
<th>Elapsed Time</th>
<th>Relative Time</th>
<th>User Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim1_3_3_arma()</td>
<td>100</td>
<td>2.118</td>
<td>1.00000</td>
<td>2.12</td>
</tr>
<tr>
<td>sim1_3_3_rcpp()</td>
<td>100</td>
<td>2.192</td>
<td>1.03494</td>
<td>2.19</td>
</tr>
<tr>
<td>sim1_3_3_R()</td>
<td>100</td>
<td>153.772</td>
<td>72.60246</td>
<td>153.70</td>
</tr>
<tr>
<td>sim1_3_3_Rcomp()</td>
<td>100</td>
<td>154.251</td>
<td>72.82861</td>
<td>154.19</td>
</tr>
</tbody>
</table>
```

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

Suggestions (by Albert): replace \( n, m \), standard deviations of Normal RNG, replace Normal RNG, ... which, thanks to **Rcpp** and ’Rcpp sugar’ is a snap.
Albert reports this table:

<table>
<thead>
<tr>
<th>Populations</th>
<th>True Sign. Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal pop. with equal spreads</td>
<td>0.0986</td>
</tr>
<tr>
<td>Normal pop. with unequal spreads</td>
<td>0.1127</td>
</tr>
<tr>
<td>$t(4)$ distr. with equal spreads</td>
<td>0.0968</td>
</tr>
<tr>
<td>Expon. pop. with equal spreads</td>
<td>0.1019</td>
</tr>
<tr>
<td>Normal + exp. pop. with unequal spreads</td>
<td>0.1563</td>
</tr>
</tbody>
</table>

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

Our simulations are $\approx$ 70-times faster, so we can increase the number of simulation by 100 and reduce the standard error to $\sqrt{0.1 \times 0.9/1,000,000} = 0.0003$. 
Want to learn more?

- The **Rcpp** package has eight pdf vignettes, and numerous help pages; other packages tend to have vignettes as well.
- 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 O insied
Apr 8, 2013 » Dynamic Wrapping and Recursion with Rcpp — Kevin Ushey
Mar 14, 2013 » Using bigmemory with Rcpp — Michael Kents
Mar 12, 2013 » Generating a multivariate gaussian distribution using RcppArmadillo — Ahmadou Dicko
Feb 27, 2013 » Fast factor generation with Rcpp — Kevin Ushey

Dirk Eddelbuettel

Rcpp Workshop — Part IV: Applications
Expected May 2013.

Real Soon Now.