



ETC4500/ETC5450 Advanced R programming

Week 11: Rewriting R code in C++



Outline

- 1 Motivation
- The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

About me

Tomasz Woźniak

- senior lecturer in econometrics at the unimelb
- econometrician: Bayesian time series analyst
- develops methods for applied macro research
- loves cycling, yoga, books, volunteering, contemporary theatre, music, and arts
- I am nice!

About me

Tomasz Woźniak

- **R** enthusiast and specialised user for 17 years
- associate editor of the R Journal
- author of R packages bsvars and bsvarSIGNs



Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

Motivations

- Compiled code written in C++ runs much faster than interpreted code in R
- Coding in C++ for R applications has always been possible
- It requires:
 - writing C++ code
 - compiling it, and
 - linking it to R
- Difficulties:
 - tedious object-oriented programming
 - necessity of assuring object compatibility
 - Benefits are great, but the cost was too high

Motivations

- **Rcpp** is a family of packages by Dirk Eddelbuetel et al. facilitating the application of **C++** in **R**
- An interface for communication between **R** and **C++**
- Greatly simplifies the workflow
- Easier to benefit from the best of the two worlds:
 - C++ programs are pre-compiled assuring fast computations perfect for writing functions
 - R code is interpreted and dynamic: perfect for data analysis

Objectives for this session

- to facilitate working with C++ in R applications
- to perform a sequence of exercises
- to focus on:
 - basic programming structures
 - functional programming
 - object types: scalars, vectors, matrices, lists, etc.
 - linear algebra
 - statistical distributions

Materials for this session

- Lecture slides
- **C++** scripts:
 - nicetry.cpp
 - ▶ nicelr.cpp
 - nicelist.cpp
 - nicerig2.cpp

Learning resources

- This session!
- vignettes: for packages Rcpp and RcppArmadillo
- online resources:
 - Armadillo library documentation
 - RcppGallery
 - stackoverflow.com tag:rcpp
- François, R., Optimizing R Code with Rcpp on datacamp
- Tsuda, M., Rcpp for everyone
- Eddelbuettel, D., Seamless R and C++ Integration with Rcpp

Outline

- 1 Motivation
- The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

The first steps with Rcpp

Consider the following **C++** applications in **R**:

- Define a C++ function in an R script
 - promptly available for fast computations
- Develop a C++ function in a .cpp file
 - perfect for developing, testing, and benchmarking
- Use a function from a *.cpp file in **R** computations
 - perfect for elaborate projects
- Develop an R package using C++ code
 - perfect for sharing your work with the community

Define a C++ function in an R script

```
Rcpp::cppFunction('
  DataFrame nicetry (int n) {
    NumericVector v = rnorm(n);
    IntegerVector x = seq_len(n);
    LogicalVector y = v > 0;
    CharacterVector z(n, "nice");
    return DataFrame::create(_["v"] = v, _["x"] = x, _["y"] = y, _["z"] = z);
  }
')
nicetry(2)
```

```
v x y z
1 1.289 1 TRUE nice
2 0.427 2 TRUE nice
```

Develop a C++ function in a nicetry.cpp file

A *.cpp file sample contents:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
List nicetry (int n) {
  NumericVector v = rnorm(n);
  IntegerVector x = sea len(n):
  Logical Vector y = v > 0;
  CharacterVector z(n, "nice");
  return List::create( ["v"] = v, ["x"] = x, ["v"] = v, ["z"] = z);
/*** R
nicetrv(2)
*/
```

Develop a C++ function in a nicetry.cpp file

The script includes:

■ Rcpp library and namespace declarations (skip: Rcpp::)

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

Rcpp marker to export the nicetry function to R

```
// [[Rcpp::export]]
```

sample R script

```
/*** R
nicetry(2)
*/
```

Develop a C++ function in a nicetry.cpp file

The script includes:

the function definition

Develop a C++ function in a .cpp file

- Your turn!
- Develop a C++ function that creates a Tx3 matrix with:
 - an integer T as the only argument
 - a constant term column: NumericVector i(n, 1.0);
 - \blacksquare a linear trend $t \bar{t}$ column
 - \blacksquare a quadratic trend $(t \bar{t})^2$ column
- where t goes from 1 to T, and \bar{t} is the mean of sequence t.
 - create NumericVectors and assemble as NumericMatrix
 - use functions cumsum, mean, pow, and cbind.

Use a function from a nicelist.cpp file in R

nicelist.cpp file contents:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
List nicelist (int n) {
 NumericVector p = rnorm(n);
 NumericVector s(n):
 for (int i=0; i<n; i++) {
    s[i] = pow(p[i], 2);
 return List::create(_["p"] = p, _["s"] = s);
```

Use a function from a nicelist.cpp file in R

■ R script using the function from nicelist.cpp:

```
Rcpp::sourceCpp("nicelist.cpp")
nicelist(3)

$p
[1] -0.742 -0.987 -1.459

$s
[1] 0.550 0.974 2.127
```

Develop a C++ function in a .cpp file



Consider a Gaussian random walk:

$$y_t = y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, 1), \qquad y_0 = 0$$

Develop a **C++** function that:

- \blacksquare has an integer \top as the only argument
- returns a T-vector with Gaussian random walk

Hint: use functions rnorm and cumsum.

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

Some stats with RcppArmadillo

- Data objects from Rcpp have limited functionality
- **Armadillo** is a **C++** library for linear algebra that
 - provides a rich set of functions
 - has a simple and intuitive syntax
 - includes fast linear algebra routines, and
 - fast random number generators
 - has fantastic documentation
- RcppArmadillo is a simplified interface with Armadillo
 - allows seamless integration with Rcpp
 - easily passes data between R and C++

Some stats with RcppArmadillo: IG2 distribution

Sampling random draws from an inverted gamma 2 distribution.

A positive random variable σ^2 following an inverted gamma 2 distribution with positive scale s and shape ν parameters is denoted by:

$$\sigma^2 \sim \mathsf{IG2}\left(\mathsf{s},
u\right)$$

- **11** Generate random draw *x* from $\chi^2(\nu)$
- Return $\frac{s}{x}$

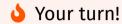


Some stats with RcppArmadillo: IG2 distribution

Contents of a nicerig2.cpp file:

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace arma;
// [[Rcpp::export]]
vec nicerig2 (
 const int n.
 const double s,
 const double nu
 vec rig2 = s / chi2rnd( nu, n );
 return rig2;
/*** R
nicerig2(2, 1, 1)
```

Develop a C++ function in a .cpp file



Consider a Gaussian random walk:

$$y_t = y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, 1), \qquad y_0 = 0$$

Develop a **C++** function using **RcppArmadillo** that:

- has an integer T as the only argument
- returns a T-vector of type vec with Gaussian random walk

Get some help HERE.

Some stats with RcppArmadillo: linear regression

Contents of a nicelr.cpp file:

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace arma;
// [[Rcpp::export]]
vec nicelr (vec v, mat x) {
  vec beta hat = solve(x.t() * x, x.t() * v);
  return beta_hat;
/*** R
x = cbind(rep(1,5),1:5); y = x %*% c(1,2) + rnorm(5)
nicelr(y, x)
*/
```

Some stats with RcppArmadillo: linear regression



Extend the nicelr function to return the covariance:

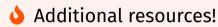
$$\widehat{Cov}\left[\widehat{\beta}\right] = \widehat{\sigma}^2 \left(X'X\right)^{-1}$$
, where $\widehat{\sigma}^2 = \frac{1}{T} \left(Y - \widehat{\beta}X\right)' \left(Y - \widehat{\beta}X\right)$

- don't adjust the arguments
- return beta_hat and cov_beta_hat in a list

Get some help HERE.

Hint: use functions inv_sympd and .n_elem.

Some stats with RcppArmadillo: Simulation smoother



Have a look at my article on Simulation Smoother using RcppArmadillo at Rcpp Gallery.

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 An R package with compiled code

Run the following code in **R**:

RcppArmadillo::RcppArmadillo.package.skeleton("nicepackage")

Note: this function has a different effect if package **pkgKitten** is installed.

DESCRIPTION includes necessary dependencies

```
Imports: Rcpp (>= 1.0.14)
LinkingTo: Rcpp, RcppArmadillo
```

NAMESPACE includes dynamic library definition and inports

```
useDynLib(nicepackage, .registration=TRUE)
importFrom(Rcpp, evalCpp)
```

- **C++** code lives in src/
 - src/Makevars files specify compilation flags
 - src/Makevars.win files specify compilation flags for Windows
 - analyse sample src/*.cpp file
 - files src/RcppExports.cpp and R/RcppExports.R are generated automatically by running Rcpp::compileAttributes()
 - analyse R wrappers to C++ functions in R/RcppExports.R



Create an **R** package with compiled code following the steps from repository donotdespair/15steps2nicepackage

- Read the README file
- download file nicepackage.R
- follow the instructions in **R**

What's next?

- Keep programming in C++ for R applications
- Reach out for help
- Read the documentation of the C++ libraries you're about to use
- Study the Rcpp family of packages
- Study openMP to facilitate parallel computing

Rewrite all your code in Rcpp! Nice!