

ETC4500/ETC5450

Advanced R programming

Week 11: Rewriting R code in C++



Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 Create an R package with compiled code in ten steps

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 Create an R package with compiled code in ten steps

About me

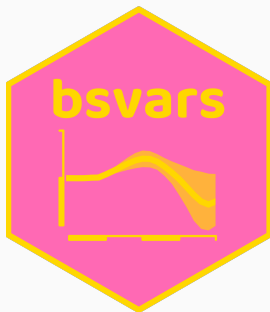
Tomasz Woźniak

- 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 art
- I am nice!

About me

Tomasz Woźniak

- **R** enthusiast and specialised user for 16 years
- **bsvars** package author (more coming up)



Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 Create an R package with compiled code in ten steps

Motivations

- 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 Eddelbuettel et al. facilitating the application of **C++** to **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++** for **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`
- **R** scripts:
 - ▶ `nicepackage.R`

learning resources

- This session!
- vignettes: for packages **Rcpp** and **RcppArmadillo**
- online resources:
 - ▶ **Armadillo** library documentation
 - ▶ RcppGallery
 - ▶ [stackoverflow.com tag:rcpp](https://stackoverflow.com/questions/tagged/rcpp)
- François, R., *Optimizing R Code with Rcpp* on datacamp
- Tsuda, M., *Rcpp for everyone*
- Eddebuettel, D., *Seamless R and C++ Integration with Rcpp*

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 Create an R package with compiled code in ten steps

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	0.259	1	TRUE	nice
2	-0.344	2	FALSE	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 = seq_len(n);
  LogicalVector y = v > 0;
  CharacterVector z(n, "nice");
  return List::create(_["v"] = v, _["x"] = x, _["y"] = y, _["z"] = z);
}

/** R
nicetry(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

```
List nicetry (                                // output type and function name
    int n                                     // input type and name
) {
    NumericVector v = rnorm(n);               // define a numeric vector and fill it
    IntegerVector x = seq_len(n);             // define an integer vector as a sequence
    LogicalVector y = v > 0;                  // define a logical vector
    CharacterVector z(n, "nice");             // define a character vector
    // return a list with the created vectors
    return List::create(_["v"] = v, _["x"] = x, _["y"] = y, _["z"] = z);
}
```

Develop a C++ function in a .cpp file

🔥 Your turn!

Develop a **C++** function that creates a $T \times 3$ matrix with:

- an integer T as the only argument
- a constant term column
- a linear trend $t - \bar{t}$ column
- a quadratic trend $(t - \bar{t})^2$ column

where t goes from 1 to T , and \bar{t} is the mean of sequence t .
Get some help [HERE](#).

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] -0.976  1.066  2.277
```

```
$s  
[1] 0.953 1.136 5.182
```

Develop a C++ function in a .cpp file

🔥 Your turn!

Consider a Gaussian random walk:

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

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

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

Outline

- 1 Motivation
- 2 The first steps with Rcpp
- 3 Some stats with RcppArmadillo
- 4 Create an R package with compiled code in ten steps

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: linear regression

Contents of a nice1r.cpp file:

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace arma;

// [[Rcpp::export]]
vec nice1r (vec y, mat x) {
  vec beta_hat = solve(x.t() * x, x.t() * y);
  return beta_hat;
}

/** R
x = cbind(rep(1,5),1:5); y = x %*% c(1,2) + rnorm(5)
nice1r(y, x)
*/
```


Some stats with RcppArmadillo: linear regression

Your turn!

Extend the `nicelr` function to return also the covariance of $\hat{\beta}$:

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

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

Get some help [HERE](#).

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:

$$\sigma^2 \sim IG2(s, \nu)$$

- 1 Generate random draw x from $\chi^2(\nu)$
- 2 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)
*/
```

Some stats with RcppArmadillo: NIG2 distribution

Normal-inverted gamma 2 distribution.

Random variables, an N -vector \mathbf{x} and a positive scalar, σ^2 , following the normal-inverted gamma 2 distribution with

- an N -vector of the mean μ
- a positive definite $N \times N$ covariance matrix Σ
- a positive scale s
- a positive shape ν

Some stats with RcppArmadillo: NIG2 distribution

Normal-inverted gamma 2 distribution.

$$(\mathbf{x}, \sigma^2) \sim NIG2(\boldsymbol{\mu}, \boldsymbol{\Sigma}, s, \nu)$$

$$p(\mathbf{x}, \sigma^2) = p(\mathbf{x} \mid \sigma^2) p(\sigma^2)$$

$$\sigma^2 \sim IG2(s, \nu)$$

$$\mathbf{x} \mid \sigma^2 \sim N(\boldsymbol{\mu}, \sigma^2 \boldsymbol{\Sigma})$$

Some stats with RcppArmadillo: NIG2 distribution

To generate n random draws from the normal-inverted gamma 2 distribution:

- 1 Generate n independent random draws of $\sigma^{2(j)}$ from $IG2(s, \nu)$ for $j = 1, \dots, n$
- 2 For each j , generate the corresponding random draw of $\mathbf{x}^{(j)}$ from $N(\boldsymbol{\mu}, \sigma^{2(j)}\boldsymbol{\Sigma})$
- 3 Return the collection of pairs $\{\mathbf{x}^{(j)}, \sigma^{2(j)}\}_{j=1}^n$

Some stats with RcppArmadillo: NIG2 distribution

Your turn!

Complement the `niceig2` function with another one that provides n random draws from the normal-inverted gamma 2 distribution.

- adjust the arguments
- return a list containing
 - ▶ an n -vector of σ^2 draws
 - ▶ an $n \times N$ matrix of \mathbf{x} draws

Get some help [HERE](#).

Some stats with RcppArmadillo: Simulation smoother

🔥 Additional resources!

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 Create an R package with compiled code in ten steps

Step 1: create a package

Run the following code in **R**:

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

- **C++** code lives in `src/`
- `DESCRIPTION` includes necessary dependencies
- `NAMESPACE` includes `useDynLib(nicepackage)`
- **R** functions in `R/` refer to **C++** functions via `.call()`
- File `R/RcppExports.R` contains all **C++** functions exported to **R**

Step 2: create R project and open it

- you know what to do

Step 3: some cleaning

■ remove files:

- ▶ Read-and-delete-me
- ▶ src/HelloWorld.cpp
- ▶ man/*

Step 4: include useful elements

- 1 Set **git** `usethis::use_git()`
- 2 Set licencing `usethis::use_gpl3_license()`
- 3 Set package doc `usethis::use_package_doc()`
- 4 Set **roxy** `roxygen2::roxygenise()`
 - copy Encoding: UTF-8 to DESCRIPTION
 - remove the NAMESPACE file
 - run `roxygen2::roxygenise()` again

Step 5: edit the DESCRIPTION file

- you know what to do!

Step 6: include compiled code

- 1 copy the `nicerig2.cpp` file to `src/` directory
- 2 edit it
- 3 include the header file `nicerig2.h` in the `src/` directory
- 4 edit it

Step 7: compile the code the first time

- 1 Run `Rcpp::compileAttributes()` to update the `R/RcppExports.R` file

Step 8: include R wrapper for C++ function

- 1 Create an R/`nicerig2.R` file
- 2 Copy/paste the corresponding function from the R/`RcppExports.R` file
- 3 Include argument check
- 4 Include documentation
- 5 Implement further adjustments

Step 9: some final touches

- 1 Run `Rcpp::compileAttributes()`
- 2 Run `devtools::document()` to generate documentation
- 3 Run `devtools::check()` to check the package

TADA! Your package is ready!

Step 10: use the package for the first time

- 1 Run `devtools::load_all()` to load the package
- 2 Run `hist(nicerig2(1e4), breaks = 100)` to test the function
- 3 Run `?nicerig2` and `?nicepackage` to check the documentation

What's next?

Rewrite all your code in Rcpp!

Nice!