Frequently Asked Questions about **Rcpp**

Dirk Eddelbuettel         Romain François

**Rcpp** version 0.12.10 as of March 17, 2017

Abstract

This document attempts to answer the most Frequently Asked Questions (FAQ) regarding the **Rcpp** (Eddelbuettel, François, Allaire, Ushey, Kou, Russel, Chambers, and Bates, 2017; Eddelbuettel and François, 2011; Eddelbuettel, 2013) package.

Contents

1 Getting started 2
   1.1 How do I get started? 2
   1.2 What do I need? 2
   1.3 What compiler can I use? 3
   1.4 What other packages are useful? 3
   1.5 What licenses can I choose for my code? 3

2 Compiling and Linking 4
   2.1 How do I use **Rcpp** in my package? 4
   2.2 How do I quickly prototype my code? 4
      2.2.1 Using inline 4
      2.2.2 Using Rcpp Attributes 4
   2.3 How do I convert my prototyped code to a package? 5
   2.4 How do I quickly prototype my code in a package? 5
   2.5 But I want to compile my code with R CMD SHLIB! 5
   2.6 But R CMD SHLIB still does not work! 6
   2.7 What about LinkingTo? 6
   2.8 Does **Rcpp** work on windows? 6
   2.9 Can I use **Rcpp** with Visual Studio? 6
   2.10 I am having problems building Rcpp on OS X, any help? 6
      2.10.1 Lack of a Compiler 6
      2.10.2 Differing Mac OS X R Versions Leading to Binary Failures 7
      2.10.3 No OpenMP Support 7
      2.10.4 Additional Information / Help 7
   2.11 Does **Rcpp** work on solaris/suncc? 8
   2.12 Does **Rcpp** work with Revolution R? 8
   2.13 Is it related to CXXR? 8
   2.14 How do I quickly prototype my code using Attributes? 8
   2.15 What about the new 'no-linking' feature? 8
   2.16 I am having problems building RcppArmadillo on OS X, any help? 8
      2.16.1 Fixed set of **gfortran** binaries 8
      2.16.2 Pre-existing or latest **gfortran** binaries 9
1 Getting started

1.1 How do I get started?

If you have Rcpp installed, please execute the following command in R to access the introductory vignette (which is a variant of the Eddelbuettel and François (2011) paper) for a detailed introduction, ideally followed by at least the Rcpp Attributes (Allaire, Eddelbuettel, and François, 2015) vignette:

```r
> vignette("Rcpp-introduction")
> vignette("Rcpp-attributes")
```

If you do not have Rcpp installed, these documents should also be available whereever you found this document, i.e., on every mirror of CRAN site.

1.2 What do I need?

Obviously, R must be installed. Rcpp provides a C++ API as an extension to the R system. As such, it is bound by the choices made by R and is also influenced by how R is configured.

In general, the standard environment for building a CRAN package from source (particularly when it contains C or C++ code) is required. This means one needs:

- a development environment with a suitable compiler (see below), header files and required libraries;
- R should be built in a way that permits linking and possibly embedding of R; this is typically ensured by the `--enable-shared-lib` option;
- standard development tools such as make etc.
Also see the RStudio documentation on pre-requisites for R package development.

1.3 What compiler can I use?

On almost all platforms, the GNU Compiler Collection (or gcc, which is also the name of its C language compiler) has to be used along with the corresponding g++ compiler for the C++ language. A minimal suitable version is a final 4.2.* release; earlier 4.2.* were lacking some C++ features (and even 4.2.1, still used on OS X as the last gcc release), has issues).

Generally speaking, the default compilers on all the common platforms are suitable.

Specific per-platform notes:

Windows users need the Rtools package from the site maintained by Duncan Murdoch which contains all the required tools in a single package; complete instructions specific to Windows are in the ‘R Administration’ manual (R Core Team, 2015a, Appendix D). As of August 2014, it still installs the gcc/g++ 4.6.* compiler which limits the ability to use modern C++ standards so only s-std=c++0x is supported. R 3.1.0 and above detect this and set appropriate flags.

OS X users, as noted in the ‘R Administration’ manual (R Core Team, 2015a, Appendix C.4), need to install the Apple Developer Tools (e.g., Xcode (OS X ≤ 10.8) or Xcode Command Line Tools (OS X ≥ 10.9) (as well as gfortran if R or Fortran-using packages are to be built); also see FAQ 2.10 and FAQ 2.16 below. Depending on whether on OS X release before or after Mavericks is used, different additional installation may be needed. Consult the r-sig-mac list (and its archives) for (current) details.

Linux users need to install the standard development packages. Some distributions provide helper packages which pull in all the required packages; the r-base-dev package on Debian and Ubuntu is an example.

The clang and clang++ compilers from the LLVM project can also be used. On Linux, they are inter-operable with gcc et al. On OS X, they are unfortunately not ABI compatible. The clang++ compiler is interesting as it emits much more comprehensible error messages than g++ (though g++ 4.8 and 4.9 have caught up).

The Intel icc family has also been used successfully as its output files can also be combined with those from gcc.

1.4 What other packages are useful?

Additional packages that we have found useful are

**inline** which is invaluable for direct compilation, linking and loading of short code snippets—but now effectively superseded by the Rcpp Attributes (see FAQ 2.2.2 and FAQ 2.14) feature provided by Rcpp;

**RUnit** is used for unit testing; the package is recommended and will be needed to re-run some of our tests but it is not strictly required during use of Rcpp;

**rbenchmark** to run simple timing comparisons and benchmarks; it is also recommended but not required.

**microbenchmark** is an alternative for benchmarking.

**devtools** can help the process of building, compiling and testing a package but it too is entirely optional.

1.5 What licenses can I choose for my code?

The Rcpp package is licensed under the terms of the GNU GPL 2 or later, just like R itself. A key goal of the Rcpp package is to make extending R more seamless. But by linking your code against R (as well as Rcpp), the combination is bound by the GPL as well. This is very clearly stated at the FSF website:

Linking a GPL covered work statically or dynamically with other modules is making a combined work based on the GPL covered work. Thus, the terms and conditions of the GNU General Public License cover the whole combination.

So you are free to license your work under whichever terms you find suitable (provided they are GPL-compatible, see the FSF site for details). However, the combined work will remain under the terms and conditions of the GNU General Public License. This restriction comes from both R which is GPL-licensed as well as from Rcpp and whichever other GPL-licensed components you may be linking against.

3
2 Compiling and Linking

2.1 How do I use Rcpp in my package?

Rcpp has been specifically designed to be used by other packages. Making a package that uses Rcpp depends on the same mechanics that are involved in making any R package that use compiled code — so reading the Writing R Extensions manual (R Core Team, 2015b) is a required first step. Further steps, specific to Rcpp, are described in a separate vignette.

> vignette("Rcpp-package")

2.2 How do I quickly prototype my code?

There are two toolchains which can help with this:

• The older one is provided by the inline package and described in Section 2.2.1.
• Starting with Rcpp 0.10.0, the Rcpp Attributes feature (described in Section 2.2.2) offered an even easier alternative via the function evalCpp, cppFunction and sourceCpp.

The next two subsections show an example each.

2.2.1 Using inline

The inline package (Sklyar, Murdoch, Smith, Eddelbuettel, and François, 2015) provides the functions cfunction and cxxfunction. Below is a simple function that uses accumulate from the (C++) Standard Template Library to sum the elements of a numeric vector.

```r
> fx <- cxxfunction(signature(x = "numeric"),
+ "NumericVector xx(x);
+ return wrap(std::accumulate(xx.begin(), xx.end(), 0.0));",
+ plugin = "Rcpp")
> res <- fx(seq(1, 10, by=0.5))
> res
[1] 104.5
```

One might want to use code that lives in a C++ file instead of writing the code in a character string in R. This is easily achieved by using readLines:

```r
> fx <- cxxfunction(signature(), paste(readLines("myfile.cpp"), collapse="\n"),
+ plugin = "Rcpp")
```

The verbose argument of cxxfunction is very useful as it shows how inline runs the show.

2.2.2 Using Rcpp Attributes

Rcpp Attributes (Allaire et al., 2015), and also discussed in FAQ 2.14 below, permits an even easier route to integrating R and C++. It provides three key functions. First, evalCpp provide a means to evaluate simple C++ expression which is often useful for small tests, or to simply check if the toolchain is set up correctly. Second, cppFunction can be used to create C++ functions for R use on the fly. Third, RcppsourceCpp can integrate entire files in order to define multiple functions.

The example above can now be rewritten as:

```r
> cppFunction('double accu(NumericVector x) {
+ return(std::accumulate(x.begin(), x.end(), 0.0));
}'),
> res <- accu(seq(1, 10, by=0.5))
> res
[1] 104.5
```
The `cppFunction` parses the supplied text, extracts the desired function names, creates the required scaffolding, compiles, links and loads the supplied code and makes it available under the selected identifier.

Similarly, `sourceCpp` can read in a file and compile, link and load the code therein.

### 2.3 How do I convert my prototyped code to a package?

Since release 0.3.5 of `inline`, one can combine FAQ 2.2.1 and FAQ 2.1. See `help("package.skeleton-methods")` once `inline` is loaded and use the skeleton-generating functionality to transform a prototyped function into the minimal structure of a package. After that you can proceed with working on the package in the spirit of FAQ 2.1.

`Rcpp Attributes` (Allaire et al., 2015) also offers a means to convert functions written using `Rcpp Attributes` into a function via the `compileAttributes` function; see the vignette for details.

### 2.4 How do I quickly prototype my code in a package?

The simplest way may be to work directly with a package. Changes to both the R and C++ code can be compiled and tested from the command line via:

```
$ R CMD INSTALL mypkg && Rscript --default-packages=mypkg -e 'someFunctionToTickle(3.14)'
```

This first installs the packages, and then uses the command-line tool `Rscript` (which ships with `R`) to load the package, and execute the R expression following the `-e` switch. Such an expression can contain multiple statements separated by semicolons. `Rscript` is available on all three core operating systems.

On Linux, one can also use `r` from the `littler` package by Horner and Eddelbuettel which is an alternative front end to R designed for both `#!` (hashbang) scripting and command-line use. It has slightly faster start-up times than `Rscript`; and both give a guaranteed clean slate as a new session is created.

The example then becomes

```
$ R CMD INSTALL mypkg && r -l mypkg -e 'someFunctionToTickle(3.14)'
```

The `-l` option calls `suppressMessages(library(mypkg))` before executing the R expression. Several packages can be listed, separated by a comma.

More choice are provide by the `devtools` package, and by using RStudio. See the respective documentation for details.

### 2.5 But I want to compile my code with `R CMD SHLIB`!

The recommended way is to create a package and follow FAQ 2.1. The alternate recommendation is to use `inline` and follow FAQ 2.2.1 because it takes care of all the details.

However, some people have shown that they prefer not to follow recommended guidelines and compile their code using the traditional `R CMD SHLIB`. To do so, we need to help `SHLIB` and let it know about the header files that `Rcpp` provides and the C++ library the code must link against.

On the Linux command-line, you can do the following:

```
$ export PKG_LIBS='Rscript -e "Rcpp:::LdFlags()"'  # if Rcpp older than 0.11.0
$ export PKG_CXXFLAGS='Rscript -e "Rcpp:::CxxFlags()"'
$ R CMD SHLIB myfile.cpp
```

which first defines and exports two relevant environment variables which `R CMD SHLIB` then relies on. On other operating systems, appropriate settings may have to be used to define the environment variables.

This approach corresponds to the very earliest ways of building programs and can still be found in some deprecated documents (as e.g. some of Dirk’s older ‘Intro to HPC with R’ tutorial slides). It is still not recommended as there are tools and automation mechanisms that can do the work for you.

`Rcpp` versions 0.11.0 or later can do with the definition of `PKG_LIBS` as a user-facing library is no longer needed (and hence no longer shipped with the package). One still needs to set `PKG_CXXFLAGS` to tell R where the `Rcpp` headers files are located.
Once R CMD SHLIB has created the dynamically-loadable file (with extension .so on Linux, .dylib on OS X or .dll on Windows), it can be loaded in an R session via dyload, and the function can be executed via .Call. Needless to say, we strongly recommend using a package, or at least Rcpp Attributes as either approach takes care of a lot of these tedious and error-prone manual steps.

2.6 But R CMD SHLIB still does not work!

We have had reports in the past where build failures occurred when users had non-standard code in their ~/.Rprofile or Rprofile.site (or equivalent) files.

If such code emits text on stdout, the frequent and implicit invocation of Rscript -e "..." (as in FAQ 2.5 above) to retrieve settings directly from Rcpp will fail.

You may need to uncomment such non-standard code, or protect it by wrapping it inside if (interactive()), or possibly try to use Rscript --vanilla instead of plain Rscript.

2.7 What about LinkingTo?

R has only limited support for cross-package linkage.

We now employ the LinkingTo field of the DESCRIPTION file of packages using Rcpp. But this only helps in having R compute the location of the header files for us.

The actual library location and argument still needs to be provided by the user. How to do so has been shown above, and we recommend you use either FAQ 2.1 or FAQ 2.2.1 both which use the Rcpp function Rcpp:::LdFlags().

If and when LinkingTo changes and lives up to its name, we will be sure to adapt Rcpp as well.

An important change arrive with Rcpp release 0.11.0 and concern the automatic registration of functions; see Section 2.15 below.

2.8 Does Rcpp work on windows?

Yes of course. See the Windows binaries provided by CRAN.

2.9 Can I use Rcpp with Visual Studio?

Not a chance.

And that is not because we are meanies but because R and Visual Studio simply do not get along. As Rcpp is all about extending R with C++ interfaces, we are bound by the available toolchain. And R simply does not compile with Visual Studio. Go complain to its vendor if you are still upset.

2.10 I am having problems building Rcpp on OS X, any help?

There are three known issues regarding Rcpp build problems on OS X. If you are building packages with RcppArmadillo, there is yet another issue that is addressed separately in FAQ 2.16 below.

2.10.1 Lack of a Compiler

By default, OS X does not ship with an active compiler. To enable a compiler one must either install Xcode (OS X ≤ 10.8) or Xcode Command Line Tools (OS X ≥ 10.9). We will focus on the later as the installation requires the use of Terminal and the install size is significantly less than the prior, which is setup using an installer.

To install XCode Command Line Tools, one must do the following:

1. Open Terminal found in /Applications/Utilities/
2. Type the following:

   $ xcode-select --install

3. Press "Install" on the window that pops up.
4. After the installation is complete, type the following in Terminal to ensure the installation was successful:

```
$ gcc --version
```

After major system updates, e.g. going from version 10.11 to 10.12, you may need to accept the terms and licenses associated the the Xcode command line tools prior to being allowed to compile again.

To do so, open the Terminal found in /Applications/Utilities/ and type:

```
$ git
```

Press spacebar to move down to the end of the file. There, you should see a prompt asking whether or not you accept the terms via either "Yes" or "No". Enter "Yes" if you agree to the terms to have the command line tools reactivated.

### 2.10.2 Differing Mac OS X R Versions Leading to Binary Failures

There are currently two distinct versions of R for OS X. The first version is a legacy version meant for Mac OS X 10.6 (Snow Leopard) - 10.8 (Mountain Lion). The second version is for more recent system Mac OS X 10.9 (Mavericks), 10.10 (Yosemite), 10.11 (El Capitan). The distinction comes as a result of a change in the compilers shipped with the operating system. As a result, avoid sending package binaries if it is known that your collaborators are working on older systems as the R binaries for these two versions will not be able to mix.

### 2.10.3 No OpenMP Support

The OS X operating environment lacks the ability to parallelize sections of code using the OpenMP standard. As a result, make sure to protect any reference to OpenMP. In this case, protect the inclusion of headers with:

```c
#ifdef _OPENMP
    #include <omp.h>
#endif
```

And when one goes to parallelize portions of code use:

```c
#ifdef _OPENMP
    // multithreaded OpenMP version of code
#else
    // single-threaded version of code
#endif
```

Doing so will enable the parallelization of the process on Linux and Windows. In the event that Apple enables OpenMP later on, this code will also allow for parallelization to occur.

The reason for the lack of OpenMP support is because under OS X, you are not using the gcc compiler. Instead, all the requests are being redirected to llvm. As of LLVM 3.7, the community initiative to enable OpenMP has been merged into the official branch. Thus, there is hope in the next release of Xcode (around WWDC in June 2016) that OpenMP will work on OS X.

### 2.10.4 Additional Information / Help

Below are additional resources that provide information regarding compiling Rcpp code on OS X.

1. A helpful post was provided by Brian Ripley regarding the use of compiling R code with OS X in April 2014 on the r-sig-mac list, which is generally recommended for OS X-specific questions and further consultation.
2. Another helpful write-up for installation / compilation on OS X Mavericks is provided by the BioConductor project.
3. Lastly, another resource that exists for installation / compilation help is provided at [http://thecoolasasprofessor.com/programming/r-compiler-tools-for-rcpp-on-os-x/](http://thecoolasasprofessor.com/programming/r-compiler-tools-for-rcpp-on-os-x/).

**Note:** If you are running into trouble compiling code with RcppArmadillo, please also see FAQ 2.16 listed below.
2.11 Does Rcpp work on solaris/suncc?
Yes, it generally does. But as we do not have access to such systems, some issues persist on the CRAN test systems.

2.12 Does Rcpp work with Revolution R?
We have not tested it yet. Rcpp might need a few tweaks to work with the compilers used by Revolution R (if those differ from the defaults).

2.13 Is it related to CXXR?
CXXR is an ambitious project that aims to totally refactor the R interpreter in C++. There are a few similarities with Rcpp but the projects are unrelated.
CXXR and Rcpp both want R to make more use of C++ but they do it in very different ways.

2.14 How do I quickly prototype my code using Attributes?
Rcpp version 0.10.0 and later offer a new feature 'Rcpp Attributes' which is described in detail in its own vignette (Allaire et al., 2015). In short, it offers functions evalCpp, cppFunction and sourceCpp which extend the functionality of the cxxfunction function.

2.15 What about the new 'no-linking' feature??
Starting with Rcpp 0.11.0, functionality provided by Rcpp and used by packages built with Rcpp accessed via the registration facility offered by R (and which is used by lme4 and Matrix, as well as by xts and zoo). This requires no effort from the user / programmer, and even frees us from explicit linking instruction. In most cases, the files src/Makevars and src/Makevars.win can now be removed. Exceptions are the use of ReppArmadillo (which needs an entry PKG_LIBS=$(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)) and packages linking to external libraries they use.

But for most packages using Rcpp, only two things are required:
• an entry in DESCRIPTION such as Imports: Rcpp (which may be versioned as in Imports: Rcpp (>= 0.11.0)),
and
• an entry in NAMESPACE to ensure Rcpp is correctly instantiated, for example importFrom(Rcpp, evalCpp).
The name of the symbol does not really matter; once one symbol is imported all symbols should be available.

2.16 I am having problems building RcppArmadillo on OS X, any help?
Odds are your build failures are due to the absence of gfortran and its associated libraries. The errors that you may receive are related to either:

"-lgfortran" or "-lquadmath"

To rectify the root of these errors, there are two options available. The first option is to download and use a fixed set of gfortran binaries that are used to compile R for OS X (e.g. given by the maintainers of the OS X build). The second option is to either use pre-existing gfortran binaries on your machine or download the latest.

2.16.1 Fixed set of gfortran binaries
Within this option, you will install a pre-compiled gfortran binary from r.research.att.com/libs/. The binary listed here was compiled by Simon Urbanek the maintainer of the OS X R versions.

To install the pre-compiled gfortran binary, do the following:
1. Open Terminal found in /Applications/Utilities/
2. Type the following:

```
sudo tar fxvz gfortran-4.8.2-darwin13.tar.bz2 -C /
```
For more information on this error, please see TheCoatlessProfessor's Rcpp, RcppArmadillo and OS X Mavericks "-lgfortran" and "-lquadmath" error.

2.16.2 Pre-existing or latest gfortran binaries

Most OS X users that have a pre-existing gfortran binaries or want the latest version, typically use a custom packaging solution to install gfortran; macports, homebrew, and fink are the usual suspects here. In general, we recommend using homebrew, and we provide a short set of instructions for installing gfortran below.

After installing homebrew by following the instructions here, you can install the latest version of gfortran with:

```
brew install gcc
```

Note that gfortran is available as part of the gcc 'formula' by default and cannot be downloaded separately, but one can freely use gfortran with Apple (or LLVM) clang compilers (as used by default on OS X since Mavericks).

You may need to set the FLIBS variable in your ~/.R/Makevars to point to the location of the gfortran library paths. A solution is outlined on StackOverflow, but the relevant details are copied in brief here.

In short, you want to add this entry to your ~/.R/Makevars:

```
FLIBS='gfortran -print-search-dirs | grep \^libraries: | sed s|libraries: =|| | sed s|:|--L|g | sed s|^|--L|'
```

This invocation explicitly asks and constructs the library link paths from the gfortran's reported search paths, and produces a set of paths suitable to be passed to FLIBS. R will then search these paths when attempting to locate e.g. libgfortran when compiling RcppArmadillo or other FORTRAN-dependent code.

Also see FAQ 2.10 above, and the links provided in that answer. In the event the above solution does not satisfy all the OS X build problems.

3 Examples

The following questions were asked on the Rcpp-devel mailing list, which is our preferred place to ask questions as it guarantees exposure to a number of advanced Rcpp users. The StackOverflow tag for rcpp is an alternative; that site is also easily searchable.

Several dozen fully documented examples are provided at the Rcpp Gallery – which is also open for new contributions.

3.1 Can I use templates with Rcpp?

*I'm curious whether one can provide a class definition inline in an R script and then initialize an instance of the class and call a method on the class, all inline in R.*

This question was initially about using templates with inline, and we show that (older) answer first. It is also easy with Rcpp Attributes which is what we show below.

3.1.1 Using inline

Most certainly, consider this simple example of a templated class which squares its argument:
inc <- template <typename T>
    class square : public std::unary_function<T,T> {
    public:
        T operator()( T t) const { return t*t ;}
    };
};

class square : public std::unary_function<T,T> {
    public:
        T operator()( T t) const { return t*t ;}
    };
};

cxxfunction(signature(xs="numeric", is="integer"),
             body="double x = Rcpp::as<double>(xs);
          int i = Rcpp::as<int>(is);
          square<double> sqdbl;
          square<int> sqint;
          return Rcpp::DataFrame::create(Rcpp::Named("x", sqdbl(x)),
                                           Rcpp::Named("i", sqint(i));
          ");
         
fun <- cxxfunction(signature(xs="numeric", is="integer"),
                            body="double x = Rcpp::as<double>(xs);
                     int i = Rcpp::as<int>(is);
                     square<double> sqdbl;
                     square<int> sqint;
                     return Rcpp::DataFrame::create(Rcpp::Named("x", sqdbl(x)),
                                                      Rcpp::Named("i", sqint(i));
                     ");
         
fun(2.2, 3L)

3.1.2 Using Rcpp Attributes
We can also use ‘Rcpp Attributes’ (Allaire et al., 2015)—as described in FAQ 2.2.2 and FAQ 2.14 above. Simply place the following code into a file and use sourceCpp on it. It will even run the R part at the end.

#include <Rcpp.h>

template <typename T> class square : public std::unary_function<T,T> {
    public:
        T operator()( T t) const { return t*t ;}
    };

    /// [[Rcpp::export]]
    Rcpp::DataFrame fun(double x, int i) {
        square<double> sqdbl;
        square<int> sqint;
        return Rcpp::DataFrame::create(Rcpp::Named("x", sqdbl(x)),
                                        Rcpp::Named("i", sqint(i));
    }

    
    
    
// *** R
fun(2.2, 3L)
*/

3.2 Can I do matrix algebra with Rcpp?

Rcpp allows element-wise operations on vector and matrices through operator overloading and STL interface, but what if I want to multiply a matrix by a vector, etc ...

Currently, Rcpp does not provide binary operators to allow operations involving entire objects. Adding operators to Rcpp
would be a major project (if done right) involving advanced techniques such as expression templates. We currently do not plan to go in this direction, but we would welcome external help. Please send us a design document.

However, we have developed the RcppArmadillo package (Eddelbuettel, François, and Bates, 2016; Eddelbuettel and Sanderson, 2014) that provides a bridge between Rcpp and Armadillo (Sanderson, 2010). Armadillo supports binary operators on its types in a way that takes full advantage of expression templates to remove temporaries and allow chaining of operations. That is a mouthful of words meaning that it makes the code go faster by using fiendishly clever ways available via the so-called template meta programming, an advanced C++ technique. Also, the RcppEigen package (Bates and Eddelbuettel, 2013) provides an alternative using the Eigen template library.

3.2.1 Using inline

The following example is adapted from the examples available at the project page of Armadillo. It calculates \( x' \times Y^{-1} \times z \)

```c++
// copy the data to armadillo structures
arma::colvec x = Rcpp::as<arma::colvec> (x_);
arma::mat Y = Rcpp::as<arma::mat>(Y_);
arma::colvec z = Rcpp::as<arma::colvec>(z_);

// calculate the result
double result = arma::as_scalar(arma::trans(x) * arma::inv(Y) * z);

// return it to R
return Rcpp::wrap( result );
```

If stored in a file myfile.cpp, we can use it via inline:

```r
> fx <- cxxfunction(signature(x_="numeric", Y_="matrix", z_="numeric" ),
+ paste(readLines("myfile.cpp"), collapse="\n"),
+ plugin="RcppArmadillo" )
> fx(1:4, diag(4), 1:4)
```

The focus is on the code \( x' \times Y^{-1} \times z \), which performs the same operation as the R code \( t(x) \%*% solve(Y) \%*% z \), although Armadillo turns it into only one operation, which makes it quite fast. Armadillo benchmarks against other C++ matrix algebra libraries are provided on the Armadillo website.

It should be noted that code below depends on the version 0.3.5 of inline and the version 0.2.2 of RcppArmadillo

3.2.2 Using Rcpp Attributes

We can also write the same example for use with Rcpp Attributes:

```c++
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::export]]
double fx(arma::colvec x, arma::mat Y, arma::colvec z) {
    // calculate the result
    double result = arma::as_scalar(arma::trans(x) * arma::inv(Y) * z);
    return result;
}
```

```r
fx(1:4, diag(4), 1:4)
```

#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::export]]
double fx(arma::colvec x, arma::mat Y, arma::colvec z) {
    // calculate the result
    double result = arma::as_scalar(arma::trans(x) * arma::inv(Y) * z);
    return result;
}

/*
3.3 Can I use code from the Rmath header and library with Rcpp?

*Can I call functions defined in the Rmath header file and the standalone math library for R—as for example the random number generators?*

Yes, of course. This math library exports a subset of R, but Rcpp has access to much more. Here is another simple example. Note how we have to use and instance of the RNGScope class to set and re-set the random-number generator. This also illustrates Rcpp sugar as we are using a vectorised call to rnorm. Moreover, because the RNG is reset, the two calls result in the same random draws. If we wanted to control the draws, we could explicitly set the seed after the RNGScope object has been instantiated.

```r
fx <- cxxfunction(signature(),
+ 'RNGScope();
+ return rnorm(5, 0, 100);',
+ plugin="Rcpp")
set.seed(42)
fx()
[1] 137.09584 -56.46982 36.31284 63.28626 40.42683
fx()
```

Newer versions of Rcpp also provide the actual Rmath function in the R namespace, i.e. as R::rnorm(m,s) to obtain a scalar random variable distributed as $N(m,s)$.

Using Rcpp Attributes, this can be as simple as

```r
cppFunction('Rcpp::NumericVector ff(int n) { return rnorm(n, 0, 100); }
set.seed(42)
ff(5)
[1] 137.09584  -56.46982  36.31284   63.28626   40.42683
ff(5)
set.seed(42)
runif(5, 0, 100)
[1] 137.09584  -56.46982  36.31284   63.28626   40.42683
runif(5, 0, 100)
```

This illustrates the Rcpp Attributes adds the required RNGScope object for us. It also shows how setting the seed from R affects draws done via C++ as well as R, and that identical random number draws are obtained.

3.4 Can I use NA and Inf with Rcpp?

*R knows about NA and Inf. How do I use them from C++?*

Yes, see the following example:
3.5 Can I easily multiply matrices?

Can I multiply matrices easily?

Yes, via the RcppArmadillo package which builds upon Rcpp and the wonderful Armadillo library described above in FAQ 3.2:

```r
> A <- matrix(1:9, 3, 3)
> B <- matrix(9:1, 3, 3)
> C <- mmult(A, B)
```

Armadillo supports a full range of common linear algebra operations. The RcppEigen package provides an alternative using the Eigen template library. Rcpp Attributes, once again, makes this even easier:


```cpp
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::mat mult(arma::mat A, arma::mat B) {
  return A*B;
}

R
A <- matrix(1:9, 3, 3)
B <- matrix(9:1, 3, 3)
mult(A,B)
```

which can be built, and run, from R via a simple `sourceCpp` call—and will also run the small R example at the end.

### 3.6 How do I write a plugin for inline and/or Rcpp Attributes?

*How can I create my own plugin for use by the `inline` package?*

Here is an example which shows how to it using GSL libraries as an example. This is merely for demonstration, it is also not perfectly general as we do not detect locations first—but it serves as an example:

```r
> gslrng <- '
  int seed = Rcpp::as<int>(par);
  gsl_rng_env_setup();
  gsl_rng *r = gsl_rng_alloc (gsl_rng_default);
  gsl_rng_set (r, (unsigned long) seed);
  double v = gsl_rng_get (r);
  gsl_rng_free(r);
  return Rcpp::wrap(v);
'

> plug <- Rcpp:::Rcpp.plugin.maker(
  include.before = '#include <gsl/gsl_rng.h>',
  libs = paste("-L/usr/local/lib/R/site-library/Rcpp/lib -lRcpp",
  "-Wl,-rpath,/usr/local/lib/R/site-library/Rcpp/lib",
  "-L/usr/lib -lgsl -lgslcblas -lm"))

> registerPlugin("gslDemo", plug)

> fun <- cxxfunction(signature(par="numeric"), gslrng, plugin="gslDemo")

> fun(0)
```

Here the `Rcpp` function `Rcpp.plugin.maker` is used to create a plugin 'plug' which is then registered, and subsequently used by `inline`.

The same plugins can be used by Rcpp Attributes as well.

### 3.7 How can I pass one additional flag to the compiler?

*How can I pass another flag to the `g++` compiler without writing a new plugin?*

The quickest way is to modify the return value from an existing plugin. Here we use the default one from `Rcpp` itself in order to pass the new flag `-std=c++0x`. As it does not set the `PKG_CXXFLAGS` variable, we simply assign this. For other plugins, one may need to append to the existing values instead.

```
14
```
> myplugin <- getPlugin("Rcpp")
> myplugin$env$PKG_CXXFLAGS <- "-std=c++11"
> f <- cxxfunction(signature(), settings=myplugin, body=
> + std::vector<double> x = \{ 1.0, 2.0, 3.0 \}; // fails without -std=c++0x
> + return Rcpp::wrap(x);
> + ')
> f()

For Rcpp Attributes, the attributes Rcpp::plugin() can be used. Currently supported plugins are for C++11 as well as for OpenMP.

### 3.8 How can I set matrix row and column names?

**Ok, I can create a matrix, but how do I set its row and columns names?**

Pretty much the same way as in R itself: We define a list with two character vectors, one each for row and column names, and assign this to the dimnames attribute:

```r
> src <- '
Rcpp::NumericMatrix x(2,2);
x.fill(42);                // or more interesting values
Rcpp::List dimnms =       // two vec. with static names
    Rcpp::List::create(Rcpp::CharacterVector::create("cc", "dd"),
                        Rcpp::CharacterVector::create("ee", "ff"));
// and assign it
x.attr("dimnames") = dimnms;
return(x);
'
> fun <- cxxfunction(signature(), body=src, plugin="Rcpp")
> fun()
```

The same logic, but used with Rcpp Attributes:

```r
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::List fun(void) {
Rcpp::NumericMatrix x(2,2);
x.fill(42);                // or more interesting values
Rcpp::List dimnms =       // two vec. with static names
    Rcpp::List::create(Rcpp::CharacterVector::create("cc", "dd"),
                        Rcpp::CharacterVector::create("ee", "ff"));
// and assign it
x.attr("dimnames") = dimnms;
return(x);
}
```

### 3.9 Why can long long types not be cast correctly?

That is a good and open question. We rely on the basic R types, notably integer and numeric. These can be cast to and from C++ types without problems. But there are corner cases. The following example, contributed by a user, shows that we cannot reliably cast long types (on a 64-bit machines).
```r
> BigInts <- cxxfunction(signature(),
+ 'std::vector<long> bigints;
bigints.push_back(12345678901234567LL);
bigints.push_back(12345678901234568LL);
Rprintf("Difference of %ld\n", 12345678901234568LL - 12345678901234567LL);
return wrap(bigints);', plugin="Rcpp", includes="#include <vector>"
>
> retval <- BigInts()
> stopifnot(length(unique(retval)) == 2)
```

While the difference of one is evident at the C++ level, it is no longer present once cast to R. The 64-bit integer values get cast to a floating point types with a 53-bit mantissa. We do not have a good suggestion or fix for casting 64-bit integer values: 32-bit integer values fit into integer types, up to 53 bit precision fits into numeric and beyond that truly large integers may have to converted (rather crudely) to text and re-parsed. Using a different representation as for example from the GNU Multiple Precision Arithmetic Library may be an alternative.

### 3.10 What LaTeX packages do I need to typeset the vignettes?

I would like to typeset the vignettes. What do I need?

The TeXLive distribution seems to get bigger and bigger. What you need to install may depend on your operating system. Specific per-platform notes:

**Windows** users probably want the MiKTeX. Suggestions for a more detailed walk through would be appreciated.

**OS X** users seem to fall into camps which like or do not like brew / homebrew. One suggestion was to install MacTeX but at approximately 2.5gb (as of January 2016) this is not lightweight.

**Linux** users probably want the full TeXLive set from their distribution. On Debian these packages are installed to build the R package itself: texlive-base, texlive-latex-base, texlive-generic-recommended, texlive-fonts-recommended, texlive-fonts-extra, texlive-extra-utils, texlive-latex-recommended, texlive-latex-extra. Using texlive-full may be a shortcut. Fedora and other distributions should have similar packages.

### 3.11 Why is there a limit of 20 on some constructors?

Ok, I would like to pass N object but you only allow 20. How come?

In essence, and in order to be able to compile it with the largest number of compilers, Rcpp is constrained by the older C++ standards which do not support variadic function arguments. So we actually use macros and code generator scripts to explicitly enumerate arguments, and that number has to stop at some limit. We chose 20.

A good discussion is available at this StackOverflow question concerning data.frame creation with Rcpp. One solution offers a custom ListBuilder class to circumvent the limit; another suggests to simply nest lists.

### 3.12 Can I use default function parameters with Rcpp?

Yes, you can use default parameters with some limitations. The limitations are mainly related to string literals and empty vectors. This is what is currently supported:

- String literals delimited by quotes (e.g. "foo")
- Integer and Decimal numeric values (e.g. 10 or 4.5)
- Pre-defined constants including:
  - Booleans: true and false
  - Null Values: R_NilValue, NA_STRING, NA_INTEGER, NA_REAL, and NA_LOGICAL.
- Selected vector types can be instantiated using the empty form of the ::create static member function.
  - CharacterVector, IntegerVector, and NumericVector
- Matrix types instantiated using the rows, cols constructor Rcpp::Matrix n(rows, cols)
  - CharacterMatrix, IntegerMatrix, and NumericMatrix
To illustrate, please consider the following example that provides a short how to:

```cpp
#include <Rcpp.h>

// [[Rcpp::export]]
void sample_defaults(NumericVector x = NumericVector::create(), // Size 0 vector
  bool bias = true, // Set to true
  std::string method = "rcpp rules!"){ // Default string
  Rcpp::Rcout << "x size: " << x.size() << ", ";
  Rcpp::Rcout << "bias value: " << bias << ";
  Rcpp::Rcout << "method value: " << method << std::endl;
}

/*** R
sample_defaults() # all defaults
sample_defaults(1:5) # supply x values
sample_defaults(bias = FALSE, method = "rstats") # supply bool and string
*/
```

Note: In cpp, the default bool values are true and false whereas in R the valid types are TRUE or FALSE.

### 3.13 Can I use C++11, C++14, C++17, ... with Rcpp?

But of course. In a nutshell, this boils down to what your compiler supports, and also what R supports. We expanded a little on this in Rcpp Gallery article providing more detail. What follows in an abridged summary.

You can always locally set appropriate PKG_CXXFLAGS as an environment variable, or via `.R/Makevars`. You can also plugins and/or R support from src/Makevars:

- **C++11** has been supported since early 2013 via a plugin selecting the language standard which is useful for sourceCpp() etc. For packages, R has supported it since R 3.1.0 which added the option to select the language standard via CXX_STD = CXX11. As of early 2017, over 120 packages on CRAN use this.

- **C++14** has been supported since early 2016 via a plugin selecting the language standard which is useful for sourceCpp() etc. For packages, R supports it since R 3.4.0 adding the option to select the language standard via CXX_STD = CXX14.

- **C++17** is itself more experimental now, but if you have a compiler supporting (at least parts of) it, you can use it via plugin (starting with Rcpp 0.12.10) for use via sourceCpp(), or via PKG_CXXFLAGS or other means to set compiler options. R support may be available at a later date.

### 4 Support

#### 4.1 Is the API documented?

You bet. We use doxygen to generate html, latex and man page documentation from the source. The html documentation is available for browsing, as a very large pdf file, and all three formats are also available a zip-archives: html, latex, and man.

#### 4.2 Does it really work?

We take quality seriously and have developed an extensive unit test suite to cover many possible uses of the Rcpp API. We are always on the look for more coverage in our testing. Please let us know if something has not been tested enough.
4.3 Where can I ask further questions?

The Rcpp-devel mailing list hosted at R-forge is by far the best place. You may also want to look at the list archives to see if your question has been asked before.
You can also use Stack Overflow via its ’rcpp’ tag.

4.4 Where can I read old questions and answers?

The normal Rcpp-devel mailing list hosting at R-forge contains an archive, which can be searched via swish.
Alternatively, one can also use Gmane on Rcpp-devel as well as Mail-Archive on Rcpp-devel both of which offer web-based interfaces, including searching.

4.5 I like it. How can I help?

We maintain a list of open issues in the Github repository. We welcome pull requests and suggest that code submissions come corresponding unit tests and, if applicable, documentation.
If you are willing to donate time and have skills in C++, let us know. If you are willing to donate money to sponsor improvements, let us know too.
You can also spread the word about Rcpp. There are many packages on CRAN that use C++, yet are not using Rcpp. You could blog about it, or get the word out otherwise.
Last but not least the Rcpp Gallery is open for user contributions.

4.6 I don’t like it. How can I help?

It is very generous of you to still want to help. Perhaps you can tell us what it is that you dislike. We are very open to constructive criticism.

4.7 Can I have commercial support for Rcpp?

Sure you can. Just send us an email, and we will be happy to discuss the request.

4.8 I want to learn quickly. Do you provide training courses?

Yes. Just send us an email.

4.9 Where is the code repository?

From late 2008 to late 2013, we used the Subversion repository at R-Forge which contained Rcpp and a number of related packages. It still has the full history as well as number of support files.
We have since switched to a Git repository at Github for Rcpp (as well as for RcppArmadillo and RcppEigen).

References


