
Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org

Center for Research Methods and Data Analysis
University of Kansas
November 16, 2013
Outline

1. Why?
   - R
   - C++
   - Vision
   - Features
Why R?

Programming with Data


Thanks to John Chambers for sending me high-resolution scans of the covers of his books.
Why R?
Succinct and expressive

```r
xx <- faithful[, "eruptions"]
fit <- density(xx)
plot(fit)
```

```
density.default(x = xx)
N = 272   Bandwidth = 0.3348
```
Why R?
Succinct and expressive

```r
xx <- faithful[, "eruptions"]
fit1 <- density(xx)
fit2 <- replicate(10000, {
  x <- sample(xx, replace=TRUE);
  density(x, from=min(fit1$x),
          to=max(fit1$x))$y
})
fit3 <- apply(fit2, 1,
              quantile, c(0.025, 0.975))
plot(fit1, ylim=range(fit3))
polygon(c(fit1$x, rev(fit1$x)),
        c(fit3[1,], rev(fit3[2,])),
        col='grey', border=F)
lines(fit1)
```

The example was posted by Greg Snow on r-help a few years ago.
Why R?
Extensible

R has always been extensible via

- C via a bare-bones interface described in *Writing R Extensions*
- Fortran which is also used internally by R
- Java via *rJava* by S Urbanek
- C++ but essentially at the bare-bones level of C

So 'in theory' this worked – yet tedious 'in practice'.
Why C++?

- Asking Google leads to 37,400,000 hits.
- **Wikipedia**: *C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language.*
- C++ is industrial-strength, vendor-independent, widely-used, and *still evolving*.
- In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API.
- As a widely used language it also has good tool support (debuggers, profilers, code analysis).
Why C++?
Scott Meyers: “View C++ as a federation of languages”

C provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.

Object-Oriented C++ just to provide endless discussions about exactly what OO is or should be.

Templated C++ which is mighty powerful; template meta programming unequalled in other languages.

The STL which is a specific template library which is powerful but has its own conventions.

C++11 adds enough to be called a fifth language.
Source: John Chambers, personal communication.
Why Rcpp?

**Easy to use** it really does not have to be that complicated – we will look at a few examples

**Expressive** it allows for *vectorised* C++ using *Rcpp Sugar*

**Seamless** access to all R objects: vector, matrix, list, S3/S4/RefClass, Environment, Function, ...

**Speed gains** for a variety of tasks *Rcpp* excels precisely where R struggles: loops, function calls, ...

**Extensions** greatly facilitates access to external libraries using eg *Rcpp modules* (but we will not have time for a walkthrough)
2 What?
  - R API
  - C++
What can Rcpp do?

Everything evolves around `.Call`

At the C++ level:

```cpp
SEXP foo(SEXP a, SEXP b, SEXP c, ...)
```

and at the R level:

```r
res <- .Call("foo", a, b, c, ..., 
             package="mypkg")
```
What can Rcpp do?
Seamless interchange of R objects: C API of R

#include <R.h>
#include <Rdefines.h>
SEXP convolve2(SEXP a, SEXP b) {
    int i, j, na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    PROTECT(a = AS_NUMERIC(a));
    PROTECT(b = AS_NUMERIC(b));
    na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
    PROTECT(ab = NEW_NUMERIC(nab));
    xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
    xab = NUMERIC_POINTER(ab);
    for (i = 0; i < nab; i++) xab[i] = 0.0;
    for (i = 0; i < na; i++)
        for (j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return(ab);
}
What can Rcpp do?
Seamless interchange of R objects: Rcpp version

```cpp
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector convolveCpp(NumericVector a, NumericVector b) {
  int na = a.size(), nb = b.size();
  int nab = na + nb - 1;
  NumericVector xab(nab);

  for (int i = 0; i < na; i++)
    for (int j = 0; j < nb; j++)
      xab[i + j] += a[i] * b[j];

  return xab;
}
```
What can Rcpp do?
Seamless interchange of R objects

- Any R object can be passed down to C++ code: vectors, matrices, list, ...
- But also functions, environments and more.
- This includes S3 and S4 objects as well as Reference Classes.
- Object attributes can be accessed directly.
- Objects can be created at the C++ level, and the R garbage collector *does the right thing* as if it were an R-created object.
**What can Rcpp do?**

**Seamless use of RNGs**

```r
set.seed(42); runif(5)
```

```markdown
## [1] 0.9148 0.9371 0.2861 0.8304 0.6417
```

```r
cppFunction('NumericVector r1(int n) {
    NumericVector x(n);
    for (int i=0; i<n; i++) x[i] = R::runif(0,1);
    return(x);
}
')
set.seed(42); r1(5)
```

```markdown
## [1] 0.9148 0.9371 0.2861 0.8304 0.6417
```

```r
cppFunction('NumericVector r2(int n) { return runif(n,0,1); }')
set.seed(42); r2(5)
```

```markdown
## [1] 0.9148 0.9371 0.2861 0.8304 0.6417
```
piR <- function(N) {
  x <- runif(N)
  y <- runif(N)
  d <- sqrt(x^2 + y^2)
  return(4 * sum(d <= 1.0) / N)
}
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double piSugar(const int N) {
    NumericVector x = runif(N);
    NumericVector y = runif(N);
    NumericVector d = sqrt(x*x + y*y);
    return 4.0 * sum(d <= 1.0) / N;
}

NB: Use of RNGScope ensured via Rcpp Attributes.
Outline

When?
- Example 1
- Example 2
Consider a function defined as

\[ f(n) \] such that

\[
\begin{cases}
  n & \text{when } n < 2 \\
  f(n-1) + f(n-2) & \text{when } n \geq 2
\end{cases}
\]
When do we use Rcpp?

Easy speedup: Simple R Implementation

```
fibR <- function(n) {
    if (n < 2) return(n)
    return(fibR(n-1) + fibR(n-2))
}

## Using it on first 11 arguments
sapply(0:10, fibR)
```

```
## [1] 0 1 1 2 3 5 8 13 21 34 55
```
## When do we use Rcpp?

Easy speedup: Timing R Implementation

```r
benchmark(fibR(10), fibR(15), fibR(20))[, 1:4]
```

<table>
<thead>
<tr>
<th></th>
<th>test</th>
<th>replications</th>
<th>elapsed</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fibR(10)</td>
<td>100</td>
<td>0.037</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>fibR(15)</td>
<td>100</td>
<td>0.509</td>
<td>13.76</td>
</tr>
<tr>
<td>3</td>
<td>fibR(20)</td>
<td>100</td>
<td>4.305</td>
<td>116.35</td>
</tr>
</tbody>
</table>
When do we use Rcpp?
Easy speedup: C++ Implementation

```r
cppFunction("
    int fibCpp(int n) {
        if (n < 2) return(n);
        return(fibCpp(n-1) + fibCpp(n-2));
    }
")

## Using it on first 11 arguments
sapply(0:10, fibCpp)

## [1] 0 1 1 2 3 5 8 13 21 34 55
```
When do we use Rcpp?

Easy speedup: Putting it all together

```r
fibR <- function(n) {
  if (n<2) return(n)
  return(fibR(n-1) + fibR(n-2))
}

cppFunction('int fibCpp(int n) {
  if (n<2) return n;
  return fibCpp(n-2) + fibCpp(n-1);
}');

benchmark(fibR(25), fibCpp(25), order="relative")[,1:4]
```

```markdown
<table>
<thead>
<tr>
<th>test</th>
<th>replications</th>
<th>elapsed</th>
<th>relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>fibCpp(25)</td>
<td>100</td>
<td>0.084</td>
<td>1.0</td>
</tr>
<tr>
<td>fibR(25)</td>
<td>100</td>
<td>49.286</td>
<td>586.7</td>
</tr>
</tbody>
</table>
```
Let’s consider a simple possible VAR(1) system of $k$ variables.

For $k = 2$:

$$X_t = X_{t-1}B + E_t$$

where $X_t$ is a row vector of length 2, $B$ is a 2 by 2 matrix and $E_t$ is a row of the error matrix of 2 columns.
In R code, given both the coefficient and error matrices (revealing $k$ and $n$):

```r
rSim <- function(B,E) {
  X <- matrix(0, nrow(E), ncol(E))
  for (r in 2:nrow(E)) {
    X[r,] = X[(r-1),] %*% B + E[r,]
  }
  return(X)
}
```
When do we use Rcpp?

Easy speedup: VAR(1) Simulation

cppFunction('arma::mat cppSim(arma::mat B, arma::mat E) {
    int m = E.n_rows; int n = E.n_cols;
    arma::mat X(m,n);
    X.row(0) = arma::zeros<arma::mat>(1,n);
    for (int r=1; r<m; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X;
}', depends="RcppArmadillo")

a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),ncol=2)
benchmark(cppSim(a,e),rSim(a,e),order="relative")[,1:4]

## test replications elapsed relative
## 1 cppSim(a, e) 100 0.027 1.0
## 2 rSim(a, e) 100 4.561 168.9
Sometimes speed is not the only reason

- C and C++ provide a enormous amount of libraries and APIs we may want to use
- Easy to provide access to as Rcpp eases data transfer to/from R
- *Rcpp modules* can make it even easier
Outline

4 Where?
Where is Rcpp being used?
Numbers as of late September 2013

**Rcpp** is
- used by 150 packages on CRAN
- used by another 16 package on BioConductor
- cited 77 times (Google Scholar count for 2011 paper in JSS)
Where is Rcpp being use?
Several well-known packages

Amelia  Gary King et al: Multiple Imputation; uses **Rcpp** and **RcppArmadillo**

forecast Rob Hyndman et al: (Automated) Time-series forecasting; uses **Rcpp** and **RcppArmadillo**

RStan  Andrew Gelman et al: Bayesian models / MCMC

rugarch  Alexios Ghalanos: Sophisticated financial models; using **Rcpp** and **RcppArmadillo**

lme4  Doug Bates et al: Hierarchical/Mixed Linear Models; uses **Rcpp** and **RcppEigen**.

bigviz  Hadley Wickham: High-dimensional visualization of data with 10-100 million obs.
Outline

5. How?
   - Setup
   - evalCpp
   - cppFunction
   - sourceCpp
   - skeleton
How do we use Rcpp?
Uses only standard R tools to build packages

Depending on the platform, one needs

**Windows** the Rtools kit for Windows, properly installed – see CRAN, the Installation manual and many tutorials; the **installr** package may help

**OS X** the Xcode *command-line tools* (plus possibly the Fortran compiler) – see Simon’s pages

**Linux** generally just work out of the box

Several environments can be used to work with **Rcpp** – RStudio is very popular.

No additional requirements for Rcpp beyond *being able to compile R packages.*
How do we use Rcpp?

Easy to test

```r
## evaluate a C++ expression, retrieve result
evalCpp("2 + 2")

## [1] 4

## a little fancier
evalCpp("std::numeric_limits<double>::max()")

## [1] 1.798e+308

## create ad-hoc R function 'square'
cppFunction('int square(int x) { return x*x;}')
square(7L)

## [1] 49
```
How do we use Rcpp?

**Basic Usage:** `evalCpp`

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```r
evalCpp("2 * M_PI")
```

```
## [1] 6.283
```
**How do we use Rcpp?**

**Basic Usage:** `cppFunction()`

`cppFunction()` creates, compiles and links a C++ file, and creates an R function to access it.

```r
cppFunction("
    int useCpp11() {
        auto x = 10;
        return x;
    }", plugins=c("cpp11"))

useCpp11()  # same identifier as C++ function
```

```
## [1] 10
```
sourceCpp() is the actual workhorse behind evalCpp() and cppFunction(). It is described in more detail in the package vignette Rcpp-attributes.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf RcppArmadillo, RcppEigen, RcppGSL).

We are also starting to provide plugins for other compiler features. A first plugin to enable C++11 support was added recently, as was a second one for OpenMP.
How do we use Rcpp?

Basic Usage: `Rcpp.package.skeleton()`

- To create a complete and working package, the `Rcpp.package.skeleton()` function can be used.
- It extends the base R function `package.skeleton()` and supports the same set of options.
- For Rcpp use is also supports (via additional options) *Rcpp Modules* and *Rcpp Attributes* both of which can be included with working examples.
- The vignette *Rcpp-package* has complete details.
Outline

6 What Else?
- Basics
- Gallery
- Book
The package comes with **eight pdf vignettes**, and numerous help pages.

The introductory vignettes are now **published** (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp. Stat. & Data Anal.*).

The **rcpp-devel** list is *the* recommended resource, generally very helpful, and fairly low volume.

**StackOverflow** has a fair number of posts too.

Several blog posts introduce/discuss features.
What Else?

Rcpp Gallery: 70+ working and detailed examples
What Else?
The Rcpp book

Seamless R and C++ Integration with Rcpp

In print since June 2013