Intro to Rcpp: Connecting C++ to R

Kevin Lee

Department of Statistics
Western Michigan University

January 25, 2019
Outline

1 List of Useful R Packages

2 Introduction to Rcpp
Some of the top most downloaded R packages:

1. List of Useful R Packages

2. Introduction to Rcpp
What is Rcpp?

- Sometimes R code is just not fast enough.

- We will talk about how to improve performance by rewriting key functions in C++.

- Rcpp package is a fantastic tool written by Dirk Eddelbuettel and Romain Francois.

- Rcpp makes it very simple to connect C++ to R.
Why C++?

Typical bottlenecks that C++ can address include:

- Loops that can’t be easily vectorized because subsequent iterations depend on previous ones.
- Recursive functions, or problems which involve calling functions many times.
How to Install Rcpp?

Install the latest version of Rcpp from CRAN

- `install.packages("Rcpp")`

You also need a working C++ compiler. To get it:

- On Windows, install Rtools.
- On Mac, install Xcode from the app store.
- On Linux, `sudo apt-get install r-base-dev` or similar.
Two different ways to compute $\frac{1}{1+x}$:

\begin{verbatim}
f <- function(n, x) for(i in 1:n) x <- 1/(1+x)
g <- function(n, x) for(i in 1:n) x <- (1+x)^(-1)
\end{verbatim}

Check computing time with \texttt{rbenchmark} package:

\begin{verbatim}
library(rbenchmark)
N <- 10000
benchmark(f(N,1), g(N,1), order="relative")[,1:4]
\end{verbatim}
Rcpp to compute $\frac{1}{1+x}$:

```cpp
cppFunction("double fcpp(int n, double x){
    for (int i=0; i<n; i++){
        x = 1/(1+x);
    }
    return x;
}")
```

Check computing time:

```r
benchmark(f(N,1), g(N,1), fcpp(N,1), order="relative")[,1:4]
```
R function to perform a cumulative sum on a vector:

cumsumR <- function(x){
    for (i in 2:length(x)){
        x[i] <- x[i-1] + x[i]
    }
    return(x)
}

cumsumR(1:10)
cumsum(1:10)
Key Motivation: Speed (Cumulative Sum)

Rcpp function to perform a cumulative sum on a vector:

cppFunction("NumericVector cumsumRcpp(NumericVector x){
    for (int i=1; i<x.length(); i++){  
        x[i] = x[i-1] + x[i];
    }  
    return x;
}"

cumsumRcpp(1:10)

Check computing time:

x <- c(1:10000)
benchmark(cumsumR(x), cumsumRcpp(x), order="relative")[,1:4]
R function to perform the bootstrap:

```r
bootR <- function(x, B){
  bootStatistic <- matrix(0, nrow = B, ncol = 2)
  n <- length(x)
  for(i in 1:B){
    bootSample <- x[sample(1:n, size = n, replace = TRUE)]
    bootStatistic[i, 1] <- mean(bootSample)
    bootStatistic[i, 2] <- sd(bootSample)
  }
  return(bootStatistic)
}
```

```r
set.seed(125)
dat <- rnorm(1000, mean = 21, sd = 10)
resultR <- bootR(dat, 1000)
sd(resultR[,1])
```
Rcpp function to perform the bootstrap:

```r
cppFunction("NumericMatrix bootRcpp(NumericVector x, int B){
    NumericMatrix bootStatistic(B, 2);
    int n = x.length();
    for (int i=0; i<B; i++){
        NumericVector bootSample = x[floor(runif(n, 0, n))];
        bootStatistic(i, 0) = mean(bootSample);
        bootStatistic(i, 1) = sd(bootSample);
    }
    return bootStatistic;
}")
```

```r
set.seed(125)
resultRcpp <- bootRcpp(dat, 1000)
all.equal(resultR, resultRcpp)
```
Key Motivation: Speed (Bootstrap)

Check computing time:

```r
benchmark(bootR(dat, 1000), bootRcpp(dat, 1000), order="relative")[,1:4]
```
cppFunction() allows you to write C++ functions in R:

```cpp
cppFunction("int add(int x, int y, int z) { 
    int sum = x + y + z; 
    return sum; 
}"

add(1, 2, 3)
```

When you run the above code, Rcpp will compile the C++ code and construct an R function that connects to the compiled C++ function.
Example 1

R function:
```r
one <- function(){
    1
}
```

Rcpp function:
```cpp
cppFunction("int one(){
    return 1;
}"")
```
Example 1

This function illustrates important differences between R and C++:

- The syntax to create a function looks like the syntax to call a function.
- We declare the type of output the function returns. This function returns a scalar integer.
- The scalar equivalents of numeric, integer, character, and logical vectors are: double, int, String, and bool.
- The vector equivalents are: NumericVector, IntegerVector, CharacterVector, and LogicalVector.
- We must use an explicit return statement to return a value from a function.
- Every statement is terminated by a ;.
Example 2

R function:

```r
signR <- function(x){
  if(x > 0){
    1
  } else if (x == 0){
    0
  } else{
    -1
  }
}
```

Rcpp function:

```r
cppFunction("int signC(int x){
  if(x > 0){
    return 1;
  } else if (x == 0){
    return 0;
  } else{
    return -1;
  }
}"")
```
This function illustrates difference between R and C++:

- We declare the type of each input in the same way we declare the type of the output.

This function also illustrates similarity between R and C++:

- The if statement works the same way as R’s.
- A while statement also works the same way as R’s.
Example 3

R function:
sumR <- function(x){
  n <- length(x)
  total <- 0
  for(i in 1:n){
    total <- total + x[i]
  }
  total
}

Rcpp function:
cppFunction("double sumC(NumericVector x){
  int n = x.length();
  double total = 0;
  for(int i = 0; i < n; i++){
    total += x[i];
  }
  return total;
}")
Example 3

This function illustrates difference between R and C++:

- To find the length of the vector, we use the .length() method, which returns an integer.
- The for statement has a different syntax: for(init; check; increment).
- In C++, vector indices start at 0.
- Use = for assignment, not <-
- C++ provides operators that modify in-place: total += x[i] is equivalent to total = total + x[i].
Using sourceCpp()

- Use `sourceCpp()` to load a C++ file from disk in the same way you use `source()` to load a file of R code.
- We can create a C++ file using Rstudio.
References

- Advanced R by Hadley Wickham

- Dirk Eddelbuettel website
  http://dirk.eddelbuettel.com/