

Statistics 360: Advanced R for Data Science

Lecture 10

Brad McNeney

References:

- ▶ Chapter 25 of Advanced R by Wickham.
- ▶ The Rcpp website: <http://www.rcpp.org/>
- ▶ Rcpp gallery: <https://gallery.rcpp.org/>
- ▶ Rcpp quick reference
<https://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf>

Calling C++ from R with Rcpp

- ▶ R is written in C, and so in principle it is possible to write C/C++ code that calls R and R code that calls C/C++.
 - ▶ Using R “internals” directly is complicated.
 - ▶ A base R function for passing R objects to C/C++ is `.Call`.
 - ▶ See the chapter “System and foreign language interfaces” in the “Writing R Extensions” manual.
- ▶ Rcpp provides a more user-friendly interface with C++.
 - ▶ Allows you to write C++ functions that can be called from R
 - ▶ Use it to speed up code that runs too slowly in R.
 - ▶ Of the 17345 packages on CRAN, 2385 (or more) use Rcpp
- ▶ This is a brief intro; much more info at <http://www.rcpp.org/> and elsewhere on the internet.

Use cases

- ▶ Unavoidable loops (can't avoid by vectorizing).
- ▶ Many (e.g., millions) of function calls, such as in a recursive algorithm.
 - ▶ Less overhead in C++ compared to R.
- ▶ Require advanced data structures available in a C++ library, such as the Standard Template Library(STL), but not in R.
- ▶ And many more ...

Prerequisites

- ▶ Install and load the Rcpp package

```
library(Rcpp)
```

- ▶ Install a working C++ compiler. To get it:
 - ▶ On Windows, install Rtools.
 - ▶ On Mac, install Xcode.
 - ▶ On Linux, `sudo apt-get install r-base-dev`

First example

- ▶ Example from section 25.2 of text:

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

add works like a regular R function

```
add
```

```
## function (x, y, z)  
## .Call(<pointer: 0x7f255a1814c0>, x, y, z)
```

```
add(1, 2, 3)
```

```
## [1] 6
```

- ▶ Rcpp (i) compiles the C++ code (note lag) and (ii) constructs the R function `add()` that will call this compiled code.
- ▶ The above defines a function “inline”; it is also possible to “source” C++ code (more later).

The plan

- ▶ Start simple and work up, writing functions with:
 - ▶ no inputs and a scalar output
 - ▶ scalar input and scalar output
 - ▶ vector input and scalar output
 - ▶ vector input and vector output
 - ▶ matrix input and vector output
- ▶ Keep an eye out for differences, such as the need to declare the type of objects.

No inputs, scalar output

► R:

```
one <- function() 1L
```

► Rcpp:

```
cppFunction('int one() {  
    return 1;  
}')
```

```
one()
```

```
## [1] 1
```

C++ differences

- ▶ Don't use assignment to create functions.
- ▶ Declare the type of output the function returns; e.g., a scalar integer (`int`).
- ▶ C++ has true scalars, with types `double`, `int`, `String`, and `bool`.
- ▶ A return statement is **required**
- ▶ Every statement is terminated by a `;`.

Scalar input and output

- ▶ A scalar version of the `sign()` function which returns 1, 0 or -1 for positive, zero or negative input:

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

```
cppFunction('int signC(int x) {  
  if (x > 0) {  
    return 1;  
  } else if (x == 0) {  
    return 0;  
  } else {  
    return -1;  
  }  
}')  
signC(-100)
```

```
## [1] -1
```

Notes

- ▶ In addition to declaring the type of the output, we must declare the type of the input.
- ▶ Logicals and `if-else` are the same.

Vector input and scalar output

```
sumR <- function(x) {  
  total <- 0  
  for (i in seq_along(x)) {  
    total <- total + x[i]  
  }  
  total  
}  
  
cppFunction('double sumC(NumericVector x) {  
  int n = x.size();  
  double total = 0;  
  for(int i = 0; i < n; i++) {  
    total += x[i];  
  }  
  return total;  
}')
```

```
set.seed(1)
x <- rnorm(1e5)
bench::mark(sumR(x), sumC(x), sum(x))
```

```
## # A tibble: 3 x 6
```

##	expression	min	median	`itr/sec`	mem_alloc	`gc/sec`
##	<bch:expr>	<bch:tm>	<bch:tm>	<dbl>	<bch:byt>	<dbl>
## 1	sumR(x)	2.6ms	2.7ms	358.	3.97MB	0
## 2	sumC(x)	102us	103.3us	9382.	2.49KB	0
## 3	sum(x)	76us	76.3us	12592.	0B	0

Notes

- ▶ The input (R vector) in this example is of type `NumericVector`.
 - ▶ Other R vector types are `IntegerVector`, `CharacterVector`, and `LogicalVector`.
- ▶ The `.size()` method of a vector returns the length as an integer.
- ▶ Notice the syntax of `for()`: `for(initial condn; check condn; increment)`.
 - ▶ In this case we initialise by creating variable `i` with value 0.
 - ▶ Before each iteration we check that `i < n`, and terminate if not.
 - ▶ After each iteration, increment `i` by one, using `++`.
- ▶ C++ uses zero-based indexing, `0, ... n-1` (!!!)
- ▶ `+=` increments “in-place”

Vector input, vector output

```
pdistR <- function(x, ys) { sqrt((x - ys) ^ 2) }  
cppFunction('NumericVector pdistC(double x, NumericVector ys) {  
    int n = ys.size();  
    NumericVector out(n);  
  
    for(int i = 0; i < n; ++i) {  
        out[i] = sqrt(pow(ys[i] - x, 2.0)); // pow() vs ^{}  
    }  
    return out;  
}')
```

`pdistC(10,6:15)`

```
## [1] 4 3 2 1 0 1 2 3 4 5
```

- ▶ `NumericVector out(n)` is a constructor.
- ▶ Copy an existing vector with the `clone()` function; e.g.,
`NumericVector zs = clone(ys).`

Using sourceCpp

- ▶ For functions of more than a few lines it is more convenient to define the function(s) in a source file and use sourceCpp() to link it/them to R.
- ▶ Source files must end in .cpp and include

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

- ▶ If you use the File -> New File -> C++ file feature of RStudio it will add the above lines for you.

```
sourceCpp("lec10_1.cpp") # See source file
```

```
##  
## > timesTwo(42)  
## [1] 84
```

```
timesTwo(42)
```

```
## [1] 84
```

List input, including S3 classes

- ▶ Rcpp provides wrappers for lists/data frames, functions, and attributes.
- ▶ Generally more useful for output than input, because C++ needs to know classes of list components in advance.
- ▶ For use as input, you can convert components to C++ equivalents with `as()`.
 - ▶ See the source file for the following example.

```
sourceCpp("lec10_2.cpp") # see source file
mod <- lm(mpg ~ wt, data = mtcars)
mpe(mod)
```

```
## [1] -0.01541615
```

Functions

- ▶ So far our R \leftrightarrow C++ interface has been R \rightarrow C++, but we can also call R functions from C++.
- ▶ Use type `Function` to input R functions and type `RObject` to hold general input/output.

<https://gallery.rcpp.org/articles/r-function-from-c++/>

```
sourceCpp("lec10_3.cpp") # see source file
set.seed(123)
x <- rnorm(100)
callFunction(x,fivenum)
```

```
## [1] -2.30916888 -0.49667731  0.06175631  0.69499808  2.18733299
```

```
callWithOne(function(x) x+1)
```

```
## [1] 2
```

```
fit <- lm_in_C(formula(mpg~disp),mtcars,lm)
summary(fit)$coef
```

```
##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 29.59985476 1.229719515 24.070411 3.576586e-21
## disp        -0.04121512 0.004711833 -8.747152 9.380327e-10
```

A warning from the expeRts

- ▶ From <https://gallery.rcpp.org/articles/r-function-from-c++/>
Calling a function is simple and tempting. It is also slow as there are overheads involved. And calling it repeatedly from inside your C++ code, possibly buried within several loops, is outright silly. This has to be slower than equivalent C++ code, and even slower than just the R code (because of the marshallng of data). Do it when it makes sense, and not simply because it is available.

Attributes

- ▶ Get and set R object attributes, such as name and class, with the `.attr()` method for R vector types from Rcpp.
 - ▶ Rcpp also provides a `.names()` method specifically for the names attribute.

```
sourceCpp("lec10_4.cpp") # see source file  
attribs()
```

```
## a b c  
## 1 2 3  
## attr(,"my-attr")  
## [1] "my-value"  
## attr(,"class")  
## [1] "my-class"
```


Further Reading

- ▶ Missing values, section 25.4
- ▶ Rcpp gives us access to the data structures and algorithms provided by C++ libraries like the Standard Template Library.
 - ▶ See Section 25.5 for examples.
 - ▶ See also “The Algorithm Design Manual” (<https://www.algorist.com/>), or the online text and course “Algorithms” (<https://algs4.cs.princeton.edu/home/>)
- ▶ Two case studies are given in Section 25.6
- ▶ Using Rcpp and C++ code in an R package is discussed in Section 25.7