Lecture 26: C++ and Rcpp

A snippet of C++ code in R

```
1 Rcpp::cppFunction('int add(int x, int y, int z) {
2   int sum = x + y + z;
3   return sum;
4 }')
5
6 add(1, 2, 3)
```

```
[1] 6
```

What is this code doing?

Adding 3 integers together

- Have to define the type of everything
- Need to define when the object is first created / referenced
- Defining a function: name of the function

```
int add(int x, int y, int z) {
  (body of the function)
}
```

C++ code

```
1 int add(int x, int y, int z) {
2   int sum = x + y + z;
3   return sum;
4 }
```

assignment

What are some differences between C++ and R code?

- C++: need to specify type of everything
- C++: compiled beforehand
- R allows implicit returns
- naming function:

R:

```
add <- function(x, y, z){

}
```

C++:

```
int add(int x, ...){

}
```

- C++: semicolon to end lines

C++ code

Here's another function:

```
1  int signC(int x) {
2    if (x > 0) {
3      return 1;
4    } else if (x == 0) {
5      return 0;
6    } else {
7      return -1;
8    }
9  }
```

returning the sign (positive, negative, or 0)

test for equality

What similarities do you notice between C++ and R?

- if... else... identical in C++ and R
- == test for equality

C++ code

```
1  double sumC(NumericVector x) {
2    int n = x.size();
3    double total = 0;
4    for(int i = 0; i < n; ++i) {
5      total += x[i];
6    }
7    return total;
8  }
```

vector object

decimal #s

length of x

increment

for loop

starting point

stopping point

updating total

What is this code doing?

- add all the entries in a vector

- indices in C++ start at 0
- ++i (or i++) "add 1 to i"
- += shorthand for total = total + x[i]
- x[i] ith entry in x

Comparing R and C++ speed

```
1  Rcpp::cppFunction('double sumC(NumericVector x) {
2    int n = x.size();
3    double total = 0;
4    for(int i = 0; i < n; ++i) {
5      total += x[i];
6    }
7    return total;
8  }')
9
10 x <- rnorm(1000)
11 bench::mark(
12   sum(x),
13   sumC(x)
14 )
```

Sum function in R

C++ version

```
# A tibble: 2 × 6
  expression      min   median `itr/sec` mem_alloc `gc/sec`
  <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
1 sum(x)     113.33µs 113.58µs     8701.        0B        0
2 sumC(x)      2.25µs   3.12µs   320880.    2.49KB        0
```

much faster

(~30× faster)

C++ code

```
1  NumericVector col_meansC(NumericMatrix x) {
2    int n_cols = x.ncol();
3    int n_rows = x.nrow();
4    NumericVector col_means(n_cols);
5
6    double total = 0;
7
8    for(int j = 0; j < n_cols; ++j){
9      total = 0;
10     for(int i = 0; i < n_rows; ++i){
11       total += x(i,j);
12     }
13     col_means[j] = total/n_rows;
14   }
15
16   return col_means;
17 }
```

return a vector (of column means)

input is a matrix

getting dimensions of our matrix

creating a vector (of 0s) of length n_cols

loop over columns

for each column: add up all entries (loop over rows)

divide sum by # of rows

Parentheses instead of [ ] to index matrix

Comparing R and C++ speed

```
1 x <- matrix(rnorm(1000*150), ncol=150)
2
3 bench::mark(
4   colMeans(x),
5   col_meansC(x)
6 )
```

```
# A tibble: 2 × 6
  expression          min   median `itr/sec` mem_alloc `gc/sec`
  <bch:expr>     <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
1 colMeans(x)      4.04ms   4.07ms      244.   25.45KB        0
2 col_meansC(x) 123.21µs  124.5µs     7907.    3.71KB        0
```

quite a bit faster with C++ implementation

Some key points

- C++ *always* needs to know the **type** of an object
  - This is true for inputs, outputs, *and* any variables you create
- In C++, indexing begins at `0`
- C++ needs a `;` at the end of each line
- `NumericVector` objects are the equivalent of vectors in R
- `NumericMatrix` objects are the equivalent of matrices in R

Class activity

https://sta279-f23.github.io/class_activities/ca_lecture_26.html