Lecture 26: C++ and Rcpp

A snippet of C++ code in R

load C++ code into R

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

add(1, 2, 3)
```

```
[1] 6
```

return the sum of 3 integers

return type

function

declare input types (in this case, int)

calling the function in R

What is this code doing?

Adding 3 integers together

Function: takes in 3 ints

returns an int

when we make a new object (e.g., sum) we need to specify its type

Here: we returning "sum"

if sum was not an int, we would get an error

C++ code

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

assignment

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

- C++: need to specify the type of everything
- naming function:

R:

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

}
```

C++

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

}
```

- C++: semicolon to end lines

C++ code

Here's another function:

returns an integer

input: an integer

returning the sign of an integer

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

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

- if ... else if ... else is the same in C++ and R
- == to test for equality

C++ code

vector object (behaves similarly to vectors in R)

return type double e.g. (1.0, 1.5, 1.9317..)

```
double sumC(NumericVector x) {
  int n = x.size();
  double total = 0;
  for(int i = 0; i < n; ++i) {
    total += x[i];
  }
  return total;
}
```

length of x

x.size() gives length of a NumericVector

add 1 to i

updating total with x[i]

total = total + x[i]

What is this code doing?

Sums up all entries in a vector

for loop: start at $i = 0$

stop at $i = n-1$

increment by 1 each time

$i = 0$, then $i = 1$, then $i = 2$, ...

(In R: for(i in 1:n) {...})

- In C++, indices start at 0
- $x[i]$ i^{th} entry in x

Comparing R and C++ speed

```r
Rcpp::cppFunction('double sumC(NumericVector x) {
  int n = x.size();
  double total = 0;
  for(int i = 0; i < n; ++i) {
    total += x[i];
  }
  return total;
}')

x <- rnorm(1000)
bench::mark(
  sum(x),
  sumC(x)
)
```

← Sum function in R

← our 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)     111.75µs 112.12µs     8849.        0B        0
2 sumC(x)      2.33µs   3.25µs   310597.    2.49KB        0
```

about 30 times faster

C++ code

return a vector (one entry, i.e. one column mean, per column)

input is a matrix

```
NumericVector col_meansC(NumericMatrix x) {
  int n_cols = x.ncol();
  int n_rows = x.nrow();
  NumericVector col_means(n_cols);

  double total = 0;

  for(int j = 0; j < n_cols; ++j){
    total = 0;
    for(int i = 0; i < n_rows; ++i){
      total += x(i,j);
    }
    col_means[j] = total/n_rows;
  }

  return col_means;
}
```

getting dimensions of our matrix (# cols, # rows)

create a vector (of 0s) of length n_cols

variable to store the total

iterating over columns

reset total for each new column

iterating over rows

x(i,j) value at row i, column j

getting the mean (dividing the sum by # of rows)

Comparing R and C++ speed

```r
x <- matrix(rnorm(1000*150), ncol=150)

bench::mark(
  colMeans(x),
  col_meansC(x)
)
```

```
# 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      245.   25.45KB        0
2 col_meansC(x) 123.33µs 127.38µs     7779.    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

Some useful C++ code

- `.size()` returns the length of a `NumericVector`
- `.ncol()` and `.nrow()` give the numbers of rows and columns for a `NumericMatrix`
- `a += b` is shorthand for `a = a + b`
- `pow()` is used for exponentiation (e.g., `pow(3, 2)`)
- `int` is an integer value, `double` is a decimal value

Class activity

https://sta279-s24.github.io/class_activities/ca_lecture_26.html