Integrating C(++) and R

In today's lecture, we will learn how to write and run C(++) functions in R and why this may be advantageous in some situations.

Motivations

- Recall in Lab 11, we tested an algorithm that finds prime numbers.
  - When written in C, the algorithm is around 80-90 times faster than the same algorithm written in R.
- Loops in R can be very very slow...
- Solution: **Vectorization**!
  - Vectorized code are translated to SIMD instructions by R interpreter to accelerate computation.
  - Example: matrix multiplication, `pdist`, etc.
- However, **not all operations** can be vectorized.

Motivations (Tutorial, week 11)

- Consider the gradient descent algorithm:

```
f <- function (x){
    return(sin(x)^3+x^2+1)
}

df <- function(x){
    return(3*sin(x)^2*cos(x) + 2*x)
}

x <- .5
while( abs(df(x)) > .01){
    x <- x - .1*df(x)
}
```

- This `while` loop cannot be vectorized: The current iteration depends on previous iterations.

Motivations: Pairwise distance

- The p-Minkowski distance is defined as:

$$\text{dist}(\boldsymbol{a}, \boldsymbol{b}) := \left(\sum_k |a_k - b_k|^p\right)^{\frac{1}{p}}.$$

- Consider `pdist` (A, B) function that returns a distance matrix D whose i, j-th entry is the p-Minkowski distance of the i-th row of matrix A and the j-th row of B.
- When $p = 2$, Minkowski distance is the regular Euclidean distance, which can be easily vectorized.
  - Recall the `pdist4` in Lab 13.
- For other integer p, **no** simple vectorization is available.

Motivations

- Again, loops in R are slow.
- In comparison, loops in C are just a lot faster...

Motivation: Hardware Access

- C allows programmers to directly control device memory (pointers, `malloc/free`), thus is the ideal programming language for hardware-specific tasks.
- Many hardware specific libraries only provide C programming language interfaces.
  - Graphics Processing Unit (GPU) accelerated computing.
  - General-purpose Input/Output (GPIO) on Single Board Computers.
- Sometimes, it is necessary to write functions in C++ to communicate with hardware.

Integrating C++ with R

- `Rcpp` is an R package that integrates C++ programming language with R.
- With `Rcpp`, we can write functions in C++ and call them in R.
    - We get the best of both worlds.
- However, calling C++ functions from R will incur performance overhead and should only be used when it is necessary.

Hello World

- We can directly write C++ code in an R file by using `cppFunction` function provided by `Rcpp`.

```
library("Rcpp")
cppFunction("
    void sayhello(){
        Rcout << \"Hello from C!\";
    }
")
```

  - C++ code is wrapped by quotation marks.
  - There is no need to include any header files.
  - `Rcout` is a C++ IO stream and the string after `<<` will be printed to the R console.
- Copy and paste the above code to the console, enter, then you can type `sayhello()` to see the result.

Hello World

- R interpreter now recognizes `sayhello` as a regular function, just as any regular R function:

```
> sayhello
function ()
invisible(.Call(<pointer: 0x105aa1570>))
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x7fd9e07d3578>
<environment: namespace:base>
```

- Let us make a mistake in the C++ code intentionally, and see what is going on when you use `cppFunction`.

Under the hood

```
cppFunction("
    void sayhello(){
        Rcout << \"Hello from C!\" // missing ;
    }
")
```

```
file3c83297371c4.cpp:8:33: error: expected ';' after expression
        Rcout << "Hello from C!" // missing ;
                                ^
                                ;
1 error generated.
make: *** [file3c83297371c4.o] Error 1
clang++ -std=gnu++14 -I"/usr/local/Cellar/r/4.1.2/lib/R/include" -DNDEBUG   -I"/us
r/local/lib/R/4.1/site-library/Rcpp/include" -I"/private/var/folders/cx/3py866x10r
dgszj64nqfhlkm0000gn/T/RtmpSTtxRw/sourceCpp-x86_64-apple-darwin20.6.0-1.0.8.3" -I/
usr/local/opt/gettext/include -I/usr/local/opt/readline/include -I/usr/local/opt/x
z/include -I/usr/local/include   -fPIC  -g -O2  -c file3c83297371c4.cpp -o file3c8
3297371c4.o
Error in sourceCpp(code = code, env = env, rebuild = rebuild, cacheDir = cacheDir,
  :
  Error 1 occurred building shared library.
>
```

Under the hood

- R tries to compile the code wrapped up by the double quotation mark using `clang++` compiler.
  - The compilation error message indicates we missed a `;`.
- Behind the scene `Rcpp` compiles the C++ code into the machine code and load it into the interpreter for you.
  - It means your C++ code will run at "native speed" (without interpretation).

Scalar in, Scalar out

If your function has scalar input and scalar output, you can write the function using basic C types (int/float/double)

```
cppFunction("
  //the standard normal density function
  double dnorm_c(double x) {
    return 1/sqrt(2*3.141593)*exp(-x*x/2);
  }
")
```

Call it

```
> dnorm_c(1.5)
[1] 0.1295176
# compare it with the builtin dnorm
> dnorm(1.5)
[1] 0.1295176
```

Scalar in, Scalar out

```
cppFunction("
  int is_prime(int x) {
    for (int i=2; i < x; i ++){
      if(x%i ==0)
        return 0;
    }
    return 1;
  }
")
```

```
> is_prime(7)
[1] 1
> typeof(is_prime(7))
[1] "integer"
```

Vector in, Scalar Out

If your function has a vector input and a scalar output, you need to use a pre-defined C++ class `NumericVector` as the function input type.

```
cppFunction("
  double dotprod(NumericVector a, NumericVector b) {
    double s = 0;
    for(int i= 0; i < a.size(); i++){
      // C index starting from 0!!!!
      s += a[i]*b[i];
    }
    return s;
  }
")
```

- `NumericVector` is a class predefined in `Rcpp.h` header.
- It has a `.size()` method returns the length of the vector.
- You can index `NumericVector` like an array.

Vector in, Vector Out

If your function has a vector input and a vector output, then both input and output need to be objects from `NumericVector`.

```
cppFunction("
  NumericVector dnorm2_c(NumericVector x) {
    NumericVector p(x.size());
    for(int i = 0; i < x.size(); i ++){
      p[i] = 1/sqrt(2*3.1415926535)*exp(-x[i]*x[i]/2);
    }
    return p;
  }
")
```

```
> dnorm2_c(1:4)
[1] 0.2419707245 0.0539909665 0.0044318484 0.0001338302
```

- `NumericVector p(x.size())` constructs a `NumericVector` object that has the same size as `x`.

Matrix in/out

If your function has a matrix input/output, then the input/output need to be objects from `NumericMatrix`.

```
cppFunction("
  //diagonal elements in a square matrix
  NumericVector diag_c(NumericMatrix A) {
    NumericVector d(A.rows());
    for(int i = 0; i < A.rows(); i ++){
      d[i] = A(i,i); //use (,) for matrix!!!
    }
    return d;
  }
")
```

- `A.rows()` and `A.cols()` get number of rows and columns in matrix `A` respectively.
- `NumericMatrix` is indexed by `(,)`, not `[,]`!

Standalone C++ File

- In many occasions, our C++ code is heavy: it may contain multiple functions.
- In this case, it would be cleaner if we can write our C++ code in a separate file and compile/call it from another R file.

Standalone C++ File

To do so, you need to create a new C++ file, say `main.cpp` and with two special lines at the beginning:

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

These two lines will include definitions of functions and classes (such as `NumericVector`) needed for writing C++ code.

Now write C++ code normally:

```
double f(double x){
  return pow(sin(x),3)+x*x+1;
}

double df(double x){
  return 3*sin(x)*sin(x)*cos(x) + 2*x;
}
```

Standalone C++ File

Finally, write the function that you would like to call from R

```cpp
// [[Rcpp::export]]
double gradient_descent_c(double x) {
  double d = df(x);
  while(fabs(d) > 1e-8){
    x -= .1*d;
    d = df(x);
  }
  return x;
}
```

- Use `[[Rcpp::export]]` tag to tell R that this is the function you would like to call in R.

Compile C++ File and Call

First, compile and load the C++ function into the interpreter by `Rcpp::sourceCPP` function:

```
Rcpp::sourceCpp('main.cpp')
```

Now you can call your C++ function:

```
> gradient_descent_c(1)
[1] 4.3023e-09
```

Compile C++ File and Call

Compare the computation time:

```
> gradient_descent_r(1)
[1] 4.3023e-09
> Sys.time() - t1
Time difference of 0.01103187 secs
>
> t1 <- Sys.time()
> gradient_descent(1)
[1] 4.3023e-09
> Sys.time() - t1
Time difference of 0.0004069805 secs
>
```

We get about 27 times performance boost.

- For `gradient_descent_r`, we use the same implementation provided in the tutorial.

When you shouldn't use `Rcpp`

- You should not write a lightweight task in C++ and call them repeatedly from R. This will incur huge performance overhead.

```
cppFunction("
double add1(double a) {
    return a+1;
}
")

t1 <- Sys.time()
s <- 0
for(i in 1:100){
s <- add1(s)
}
Sys.time() - t1
```

```
Time difference of 0.009456873 secs
```

When you shouldn't use `Rcpp`

```
> t1 <- Sys.time()
> s <- 0
> for(i in 1:100){
+   s <- s + 1
+ }
> Sys.time() - t1
Time difference of 0.003218889 secs
```

The native R implementation is faster.

When you shouldn't use `Rcpp`

- You should not write a function that already exists in R.
  - For example, `+,-,*,/`, `%*%`, `exp/log/sin/cos`, etc.
  - These operations have already been fully optimized utilizing SIMD.
  - Writing your own C++ functions of these operations may not give you any boost on performance while making your code less readable.

Conclusion

You can call functions written in C++ from R by using `Rcpp`.

- `Rcpp` compiles your code to the machine code and load it into R interpreter, ready to be called.
- Depending your input/output types, you may need to use `NumericVector` or `NumericMatrix` classes.
- You can either write C++ inline or in a standalone file.
- Be aware of performance overhead when calling C++ functions.

Homework

1. Install `Rcpp` package. `install.packages("Rcpp")`.
2. Run the `Hello World` example provided in the lecture slides. Install necessary software required by RStudio.

Homework

3. Write a C function `pdist5(A,B)`, which takes two input matrices: A and B, and return an output matrix D, whose i,j-th entry is the "minimum distance" between the i-th row of A and j-th row of B. The minimum distance between two K-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ is defined as

$$d(\boldsymbol{a}, \boldsymbol{b}) := \min_{k \in \{1,...,K\}} |a_k - b_k|.$$

- Hint: `NumericMatrix D(m,n);` creates a `NumericMatrix` object with m rows and n columns.

4. Call `pdist5(A,B)` in R with some matrices A and B.

Homework

5. Write an R function `pdist5_r`, that does exactly the same thing.
6. Compare the computation time of `pdist5(A,B)` and `pdist5_r(A,B)` when A and B are 500 by 500 matrices.