R Programming: Worksheet 8

1. **Basic Rcpp**

   (a) Write a function in Rcpp that performs pairwise multiplication of two numeric vectors `x` and `y`. Use `cppFunction` to compile it and try it out.

   (b) Write a function in Rcpp that takes as input an integer vector `x` of length p and a square matrix `M` of dimension n rows and n columns. Return a single `double` (i.e. real number) `z` as follows

   $$z = \sum_{i=1}^{p} \begin{cases} M_{x_i,x_i}, & 1 \leq x_i \leq n \\ M_{1,1}, & x_i < 1 \\ M_{n,n}, & n < x_i \end{cases}$$

   Be careful about 0-based and 1-based indexing in c++. You might need code like `M(x(i) - 1, x(i) - 1)` using 1-based `x` form R. Note that you can get the length of a vector in Rcpp using `x.length()` and the dimension of a matrix using `M.ncol()` and `M.nrow()`. Note also that a matrix in R has type `Rcpp::NumericMatrix`.

   (c) Write a function in Rcpp that takes as input two integer vectores `x` and `y` of length p, and a matrix `M` of dimension m rows and n columns. The entries of `x` and `y` refer to 1-based positions in the rows of M, so $1 \leq x_j \leq m$, $\forall i = 1, ..., p$, and same for y. Have your function return a numeric vector `z` where the j^{th} entry of `z` is as follows.

   $$z_j = \sum_{i=1}^{p} M_{x_i,j} \times M_{y_i,j}$$

2. **Make your own R package**

   Here we will turn R code into a package. You can either replicate the functionality of the `odder` package from the lecture, or if you're feeling adventurous, choose your favourite worksheet question, and turn the resulting R code you made into a package, for example the bootstrap hypothesis testing question from Worksheet 6 (Question 3), or the Monte Carlo example from Worksheet 2 (Question 4). You can see the `odder` package from the lectures at `https://github.com/rwdavies/odder`. The code for this worksheet included on the website can replicate the `odder` package.

   (a) Save one or more R functions in some file, and optionally, one or more Rcpp functions into a c++ file

   (b) Make a package skeleton, using either RStudio built in functionality, or using code like `Rcpp.package.skeleton`

   (c) Carefully read the screen output of the package skeleton, and understand what steps you have to do to complete your R package. Pay attention to the following

   - The DESCRIPTION file
   - The NAMESPACE file (though hopefully this should be automatic)
   - The way your functions are documented (for exported functions). Try using `roxygen2` (see this url for a nice vignette `https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html`
   - *Optional* Add some tests, using `testthat`, into files like `tests/testthat/test-something.R`

(d) Make a tarball of your package. *Optional, send it to someone seated nearby, and have them try to install and run your program!*

3. ***Optional, beyond course scope*** **Advanced Rcpp re-write**

Much like in Worksheet 5, here we have real code from Marcus Tutert, a fourth year DPhil student, as part of his thesis. This code does something useful but for practical purposes is not fast enough. Here we're going to try and use Rcpp to the whole thing up, by re-writing the entire function in Rcpp.

Your goals here are two-fold. First, to succesfully implement an Rcpp version of this code. Second, to come up with a solution that's nearly as fast, or faster, than the code I've provided. Both of these are optional and beyond the scope of this course (this is a hard function to re-write!).

We're going to start with the following example data, and the following function.

```
> nsnps <- 5
> nhaps <- 10
> nweights <- 5
> gamma_quantile_weights <- sort(sample(1:(2 * nweights), nweights))
> ref_allele_matrix <- matrix(
+     rbinom(nsnps * nhaps, 1, 0.5),
+     ncol = nsnps,
+     nrow = nhaps
+ )
> nstates <- length(gamma_quantile_weights)
> weight_matrix <- matrix(
+     sample(
+         gamma_quantile_weights,
+         size    = nhaps*nsnps,
+         replace = TRUE
+     ),
+     ncol = nsnps,
+     nrow = nhaps
+ )
```

```
> current_function <- function(
+     gamma_quantile_weights,
+     ref_allele_matrix,
+     weight_matrix,
+     row_update = 5
+ ) {
+     allele_frequencies <- matrix(
+         data = NA,
+         nrow = length(gamma_quantile_weights),
+         ncol = ncol(ref_allele_matrix)
+     )
+     for (i in 1:ncol(allele_frequencies)) {
+         weight_matrix_replicated <- matrix(
+             weight_matrix[-row_update,i],
```

```
+             nrow = length(weight_matrix[-row_update,i]),
+             ncol = length(gamma_quantile_weights),
+             byrow = FALSE
+         )
+         weight_matrix_modified <- rbind(
+             weight_matrix_replicated[1:(row_update-1),],
+             gamma_quantile_weights,
+             weight_matrix_replicated[-(1:(row_update-1)),]
+         )
+         weight_matrix_sum <- (colSums(weight_matrix_modified))
+         weight_matrix_normalized <- weight_matrix_modified %*%
+             diag(1/weight_matrix_sum)
+         allele_frequencies[, i] <- colSums(
+             weight_matrix_normalized *
+             ref_allele_matrix[,i]
+         )
+     }
+     return(allele_frequencies)
+ }
```

(a) First, get the solution I've provided up and running. This includes both some R code I wrote, to facilitate the transition of the code into Rcpp, and the Rcpp code. Confirm these implementations do the same thing as the original `current_function`. Write a function that checks that the output on the example data is the same between the `current_function`, and the provided R function I wrote `robbie_R_function`, and the Rcpp function `robbie_Rcpp_function`. Try not to peek at the R or Rcpp solutions yet!

(b) Next, benchmark the three versions of the code, using `microbenchmark`, on the three versions of the code, `current_function`, `robbie_R_function`, and `robbie_Rcpp_function`. Make the dataset more realistic of real world conditions by increasing the values of the parameters to `nsnps <- 500`, `nhaps <- 1000`, `nweights <- 100`. Decrease the `times` argument of `microbenchmark` so that the benchmark doesn't take too long to run.

(c) Now it's on you! Can you do as well, or better? You might find it helpful to first copy the R code to a new function, then walk through the code line by line interactively on a small dataset (the original `nsnps <- 5`, `nhaps <- 10` and `nweights <- 5`), to understand what it's doing, and to re-write using simpler functions that you can more easily transfer over to Rcpp, for example using for loops rather than base R operations. You will probably need to use a search engine (e.g. Google) to look up c++ commands you aren't familiar with already. In the solution for this question, I write out some hints about understanding what the function is doing, that might help re-writing it.