# R's C interface tutorial

Xiaoqian Liu

May 31, 2023



# 1 Introduction

- **WHAT is R's C interface**?

  Simply speaking, you can write a function in C, and call it from R. R has some mechanism to link with foreign languages including C, C++, and Fortran. C is a low-level language, quite fast, and very widely used (incoporated into other programming languages including about 50% of R and most of the Python Standard Library). In this tutorial, we will focus on R's interface with C.

- **WHY do we need R's C interface**?

  - Speed

    This is the main reason (I believe) people needs to write their own C code and call them from R. We will show later how much you can speed up the computation if you write a function in C compared to using plain R code.

  - Other needs

    Some R packages include C code, and you may need to read or modify their source code for your own use. It is helpful if you have some knowledge of R's C interface.

- **About this tutorial**

  The purpose of this tutorial is to give a brief introduction of R's C interface, including how to call a function written in C from R, some caveats for R's C interface, and how to build an R package with C code. You need to know R well and have some basic knowledge of C. The target audience of this tutorial are R programmers who are C beginners.

# 2 Call C functions from R

## 2.1 A toy example

Let's start with a simple function in C.

```
void test(int *n)
{ int i;
  for(i=0; i<*n; i++){
     Rprintf("Test!\n");
  }
}
```

Now we show how to call this C function in R.

**The C part:**

- Save the function in basics.c
- change the working directory if needed

  'pwd' to check the current working directory; 'cd' to change the directory.
- Compile the C code in your Terminal window or VSCode (Terminal –> New Terminal) with the command (gcc complier is needed!):

```
R CMD SHLIB basics.c
```

- Obtain the two output files: **basics.o** and **basics.so**.

**The R part:**

- Load the external C code to R by the 'dyn.load' function.

```
dyn.load(''basics.so'')
```

- To invoke this C code in our R session, we need a wrapper function and using the '.C' function in R.

```
test <- function(n){
  n <- as.integer(n)
  .C("test", n=n) # the first argument is the C function's name, followed by arguments of the C
         function.
}
```

- .C also returns a list containing the arguments which we passed into the C function.
- A way to modify it.

```
test <- function(n){
  n <- as.integer(n)
  sol <- .C("test", n=n)
}
```

# 2.2 How much can C speed up the comuputation?

In my prepared 'basics.c' files, I inclued some functions for vector-matrix and matrix-matrix mutiplications. Let's see some examples to get an initial feeling of the power of C.

```
# Example of matrix multiplication
m <- 2e3
n <- 2e3
p <- 2e3

A <- matrix(rnorm(m*n), nrow = m)
B <- matrix(rnorm(n*p), nrow = n)

# run the C code
t <- proc.time()
b_C <- mmprod_C(A, B)
t_C <- proc.time()-t
t_C[[3]]


# run the R code
t <- proc.time()
b_R <- A%*%B
t_R <- proc.time()-t
t_R[[3]]

# ! ! !  The speedup depends on your computer.
```

# 3 Caveats for R's C interface

Calling C code in R needs more carefulness than using pure R code. Here are some notes for you.

- C functions called by R must all return 'void'.
- All arguments passed to the C function must be passed as pointers.
- Make sure each argument is passed with the required type (int or double).
- Each file containing the C code must include the 'R.h' header file.

```
#include <R.h>
#==== Other commonly needed header files
#include <Rmath.h>
#include <math.h>
#include <stdio.h>
#include <R_ext/BLAS.h>
#include <stdlib.h>
```

- Callocate memory and free the memory!
- Be careful about each argument you passed into C (type, name, etc.).
- Debug is hard!

# 4 Build an R package with C code

Building an R package with C code follows the same routine as building a common R package. Some extra wrok is listed below.

- Put all C functions into one c file named as the same name with your package.
- Put the c file into a foder named as 'src'.
- Write a wrapper function for each C function you want to call. In .C function, set the argument 'PACKAGE= your pcakge name'.
- Include '@useDynLib your pcakge name' in the documentation of each wrapper function.
- In your namespace file, inlcude 'useDynLib(you pcakge name)'.
- Basic steps for building an R package

```
library(devtools)
load_all()
document()
# In the Environment pane of your R studio (typically top right), click 'build'
Check
Build Source Package
Rnstall and Restart
```

# 5 Discussion

Overall, I think R's C interface is helpful for people doing optimization in Statistics or people who care about the computation of statistical problems. Using C backend code can make your code run even ten times faster than using R code. However, you have to pay for the speedup. It requires your knowledge of C. More importantly, it requires your carefulness for writing both the C and R code. I personally think that R's C interface is not friendly for debugging. It is not like running a C program, or simply writing and running some R code. I have some experience with debugging when using R's C interface. Tips that I suggested to pay much attention to have been included into the 'Caveats for R's C interface' section.

### References


1. https://cran.r-project.org/doc/manuals/R-exts.html (https://cran.r-project.org/doc/manuals/R-exts.html)
2. https://www.biostat.jhsph.edu/~rpeng/docs/interface.pdf (https://www.biostat.jhsph.edu/~rpeng/docs/interface.pdf)