Rcpp Tutorial

Jina Park

1 Install

Install Rcpp by executing the following code.

```
install.packages("Rcpp")
```

2 Introduction

There are many methods to compile C++ in r.

1. Use sourceCpp()

   (a) Open cpp file. "File → New File → C++ File"

   (b) Write follow code and save as "example1.cpp"

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

   // [[Rcpp::export]]
   NumericVector timesTwo(NumericVector x) {
     return x * 2;
   }
   ```

   (c) In R, execute the following code. (Your working directory should be in English.)

   ```
   sourceCpp("example1.cpp")
   a = 1:4
   output1 = 2*a # true answer using R
   output2 = example1(a) # answer from Rcpp;
   ```

2. Use cppFunction()

   - you can omit #include <Rcpp.h> and using namespase Rcpp; when you use cppFunction().

- Execute the following code in R.

```r
library("Rcpp")
src1 <- 'NumericVector example1(NumericVector x){
NumericVector y;
y = 2*x;
return(y);
}'

cppFunction(src1)
example1(1:4)
```

3 Format for defining a function in Rcpp.

The following code shows the basic format for defining a Rcpp function.

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

// [[Rcpp::export]]
RETURN.TYPE FUNCTION.NAME(ARGUMENT.TYPE ARGUMENT){

    //do something

    return RETURN.VALUE;
}
```

- #include <Rcpp.h> : This sentence enables you to use classes and functions defined by the Rcpp package.
- using namespace Rcpp; : This sentence is optional. But if you did not write this sentence, you have to add the prefix Rcpp:: to specify classes and functions defined by Rcpp. (For example, Rcpp::NumericVector)
- // [[Rcpp::export]]：The function defined just below this sentence will be accessible from R. You need to attach this sentence to every function you want to use from R.
- RETURN.TYPE FUNCTION.NAME(ARGUMENT.TYPE ARGUMENT)：You need to specify data types of functions and arguments.
- return RETURN.VALUE;：return statement is mandatory if your function would return a value. However, if your function do not return a value (i.e. RETURN.TYPE is void), the return statement can be omitted.

4 Rcpp main types

R	Rcpp (scalar)	Rcpp (vector)	Rcpp (matrix)
logical	bool	LogicalVector	LogicalMatrix
integer	int	IntegerVector	IntegerMatrix
double	double	NumericVector	NumericMatrix
character	String	CharacterVector	CharactorMatrix

- There are also List and DataFrame (but prefer using List).

5 Sugar

- The neat thing about Rcpp sugar enables us to write C++ code that looks almost as compact
- More information about sugar is available at
  `http://dirk.eddelbuettel.com/code/rcpp/Rcpp-sugar.pdf`

1. Operator

   - Aritmetic operators: +, -, *, /
   - Logical operatorns: $<, ><=, >=, ==, ! =$

```
library("Rcpp")
sugar <- 'List Sugar_Ex (NumericVector x, NumericVector y){
NumericVector soma = x + y, res = x - y ;
NumericVector prod = x * y, div = x / y;
LogicalVector menor = x<y, maior = x>y;
LogicalVector igual = x==y, dif = x!=y;
return List::create(soma,div,menor,dif);
}'


cppFunction(sugar)
x = c(8,9,7,6)
y = c(1,7,2,3)
Sugar_Ex(x,y)
```

2. Function

  - is_na, seq_along, seq_len, pmin and pmax, ifelse, apply, lapply, sign, diff

3. Probability

  - beta, binom, cauchy, chisq, exp, F, gamma, hyper, lnorm, logis, norm, t, pois, unif and weibull.

```
library("Rcpp")
gamma <- 'NumericVector Ex_gamma(int n){
NumericVector x = rgamma(n,1,1);
return x;
}'

cppFunction(gamma)
Ex_gamma(5)
```

6 Example: Gibbs sampler

- R

```
gibbs_r <- function(N, thin) {
  mat <- matrix(nrow = 2, ncol = N)
  x <- y <- 0
  for (i in 1:N) {
    for (j in 1:thin) {
      x <- rgamma(1, 3, y * y + 4)
      y <- rnorm(1, 1 / (x + 1), 1 / sqrt(2 * (x + 1)))
    }
    mat[, i] <- c(x, y)
  }
  mat
}
```

- Rcpp

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
```

```
NumericMatrix gibbs_cpp(int N, int thin) {
  NumericMatrix mat(2, N);
  double x = 0, y = 0;

  for(int i = 0; i < N; i++) {
    for(int j = 0; j < thin; j++) {
      x = rgamma(1, 3, 1 / (y * y + 4))[0]; // 3rd param -> inverse
      y = rnorm(1, 1 / (x + 1), 1 / sqrt(2 * (x + 1)))[0];
    }
    mat(0, i) = x;
    mat(1, i) = y;
  }
  return(mat);
}
```

7 RcppArmadillo

- R and Armadillo integration using Rcpp Armadillo is a templated C++ linear algebra library (by Conrad Sanderson) that aims towards a good balance between speed and ease of use.

```
install.packages("RcppArmadillo")
```

- When using RcppArmadillo package in cpp file, every cpp code file must have at the top:
- The #include<RcppArmadillo.h> statement provides the Rcpp.h and armadillo.h headers with the appropriate casting magic.
- The //[[Rcpp::depends(RcppArmadillo)]] statement adds onto the compiler's search path the RcppArmadillo package directory where the armadillo.h header files are found. This approach takes advantage of Rcpp Attributes.

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
```

- Rcpp and RcppArmadillo are different.
  ex) Rcpp::NumericVector, arma::vec

- The detailed descriptions of RcppArmadillo's function are in the following url.
  `http://arma.sourceforge.net/docs.html`
  `https://thecoatlessprofessor.com/programming/cpp/common-operations-with-rcpparmadillo/`
  `https://github.com/petewerner/misc/wiki/RcppArmadillo-cheatsheet`
- Example

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

arma::vec add_two(arma::vec x){
  return x + 2;
}
```

8 Reference

- `https://teuder.github.io/rcpp4everyone_en/`
- `https://privefl.github.io/R-presentation/Rcpp.html`
- `http://dirk.eddelbuettel.com/papers/rcpp_rfinance_may2016.pdf`