Speeding up R code using Rcpp and foreach packages.

Pedro Albuquerque

Universidade de Brasília

February, 8th 2017

R is a good choice as open source and free software.

But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

R is a good choice as open source and free software.
But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

UnB

R is a good choice as open source and free software.
But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

R is a good choice as open source and free software.
But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

R is a good choice as open source and free software.
But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

R is a good choice as open source and free software.
But R has some disadvantages:

- Limited memory for big datasets.
- Inefficient standard loops.

There are some solutions for both disadvantages, especially to speed up:

- Using apply family functions and vectorized operations.
- Using parallel processing.
- Using C++ (Rcpp package).

foreach package.

Suppose we are interested in estimate a variance of a complicated function of parameters, for example, the coefficient of variation:

$$\theta = \frac{\mu}{\sigma}$$

where μ and σ are the population mean and standard deviation.

We can estimate the variance of θ using, for example, the bootstrap estimator.

foreach package.

Suppose we are interested in estimate a variance of a complicated function of parameters, for example, the coefficient of variation:

$$\theta = \frac{\mu}{\sigma}$$

where μ and σ are the population mean and standard deviation.

We can estimate the variance of θ using, for example, the bootstrap estimator.

foreach package.

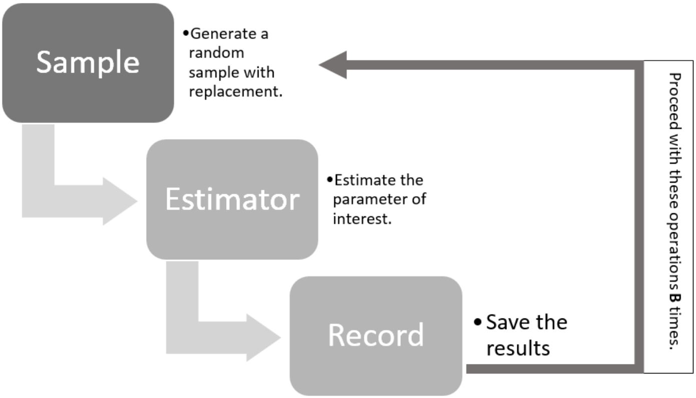
Suppose we are interested in estimate a variance of a complicated function of parameters, for example, the coefficient of variation:

$$\theta = \frac{\mu}{\sigma}$$

where μ and σ are the population mean and standard deviation.

We can estimate the variance of θ using, for example, the bootstrap estimator.

Bootstrap.


Sample
•Generate a random sample with replacement.
Estimator
•Estimate the parameter of interest.
Record
• Save the results
Proceed with these operations B times.

Bootstrap.

Bootstrap sample (b=1,...B)	Estimator
1	$\hat{\theta}_1 = \hat{\mu}_1/\hat{\sigma}_1$
2	$\hat{\theta}_2 = \hat{\mu}_2/\hat{\sigma}_2$
3	$\hat{\theta}_3 = \hat{\mu}_3/\hat{\sigma}_3$
.	
.	
B	$\hat{\theta}_B = \hat{\mu}_B/\hat{\sigma}_B$
MEAN	$\widehat{E(\theta)} = \sum \hat{\theta}_b/B$
VARIANCE	$\widehat{V(\theta)} = \sum \left(\theta_b - \widehat{E(\theta)}\right)^2/(B-1)$

Aplication.

```
#Set the seed
set.seed(1)
#Generate fake data
data <- rnorm(n=10000, mean=10, sd=10)
#True coefficient of variation
CV.true <- 10/10
#Estimated coefficient of variation
CV.hat <- mean(data)/sd(data)
```

Algorithm 1: Generating fake data.

The estimate is $\hat{\theta} \approx 0.9813$.

Aplication.

```r
#Create the bootstrap vector
result <- rep(NA, 100000)
#Start the clock
ptm <- proc.time()
for(b in 1:100000){
  #Generate the random sample with replacement
  ids <- sample(x=length(data),size=length(data),replace=T)
  sample <- data(ids)
  #Calculate the CV
  result(b) <- mean(sample)/sd(sample)
}
#Stop the clock - Time elapsed 20.40.
proc.time() - ptm
```

Algorithm 2: Time elapsed 20.40.

Aplication.

```r
library(foreach)
library(doParallel)
#See how many cores we have
ncl<-detectCores()
#Register the cores
cl <- makeCluster(ncl)
registerDoParallel(cl)
```

Algorithm 3: Using foreach package.

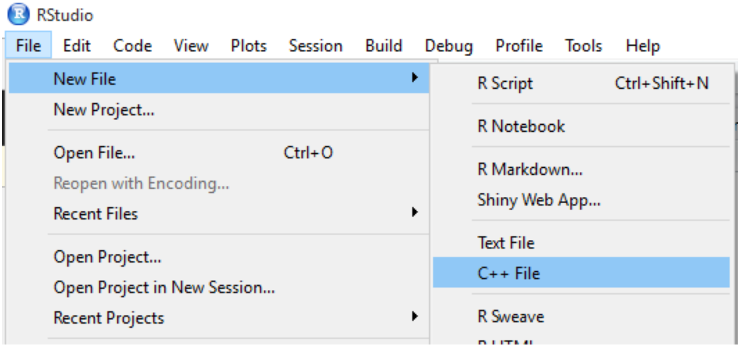
Aplication.

```r
#Start the clock
ptm <- proc.time()
#Create the bootstrap vector
result <- rep(NA, 100000)
#Bootstrap using foreach
result <- foreach(b=1:100000, .combine=c) %dopar% {
  #Generate the random sample with replacement
  ids <- sample(x=length(data),size=length(data),replace=T)
  sample <- data(ids)
  #Calculate the CV
  mean(sample)/sd(sample)
}
#Stop the clock - Time elapsed 5.32.
proc.time() - ptm
#Release the cores
stopCluster(cl)
```

Algorithm 4: Time elapsed 5.32.

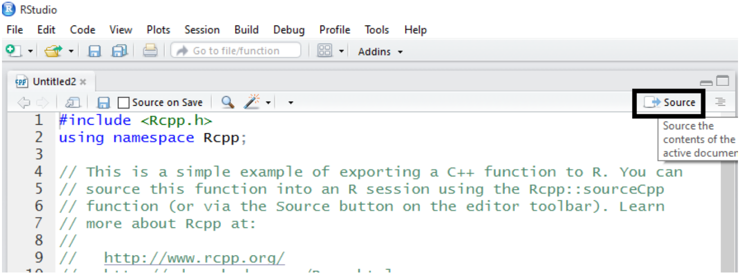
Rcpp package.

Another solution is to use the Rcpp package:



Rcpp package.

We need to compile the functions before use in R:



isPrime function.

Suppose we are interested in find if a number is prime or not.

```
#Create the isPrime function
isPrime <- function(num){
  prime <- TRUE
  den <- num -1
  while(prime==TRUE & den >1){
    #Remainder of the division
    if(num%%den==0){
      prime <- FALSE
    }
    #Decrease the number
    den <- (den - 1)
  }
  return(prime)
}
#Example isPrime(12)
```

Algorithm 5: isPrime function.

isPrime function.

Using the same idea with the Rcpp:

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
bool isPrimeCpp(int num) {
  bool prime = true;
  int den = num -1;
  while(prime==true & den >1){
  //Test if is divided
    if(num % den==0){
      prime = false;
    }
 //Decrease the number
    den = (den - 1);
  }
  return(prime);
}
```

isPrime function.

You can save your functions in a *cpp* file and invoke using:

```
#Rcpp library
library(Rcpp)
#Call the functions
sourceCpp("MyFile.cpp")
#Example
isPrimeCpp(12)
```

Algorithm 6: isPrime function.

- Create a Rcpp function to calculate the sample mean of a vector.
- Create a Rcpp function to calculate the k Fibonacci numbers.
- Use foreach and the bootstrap technique to estimate an arbitrary parameter function: $g(\theta) = \sqrt{\theta}/2$.