STAT 600 HW 1 - Performance Analysis

Kevin Korsurat

The objective of this document is to compare the performance between the functions written using Rcpp/RcppArmadillo and the base R command. We will use simple linear regression to evaluate the results. For the base R, we will utilize the `lm()` function, while the `SimpLinR()` function is implemented using Rcpp/RcppArmadillo. This document serves as the response to questions 3 and 4 in Homework 1 for STAT 600.

First, we will simulate the data. We will have 100 observations with 2 variables, denoted as x and y. We will simulate x, the predictor, from the standard normal distribution. For the response variable, y, we will generate it as $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, where $\epsilon_i \sim N(0, 1)$. We will have a total of 100 replicated data. For the purpose of reproducibility, the simulated data for this analysis is also available on GitHub under the data folder.

In this analysis, we will run the model for simple linear regression in parallel using both the `lm()` and `SimpLinR()` functions. Figure 1 shows the distribution of the estimated regression coefficients for both the `lm()` and `SimpLinR()` functions.

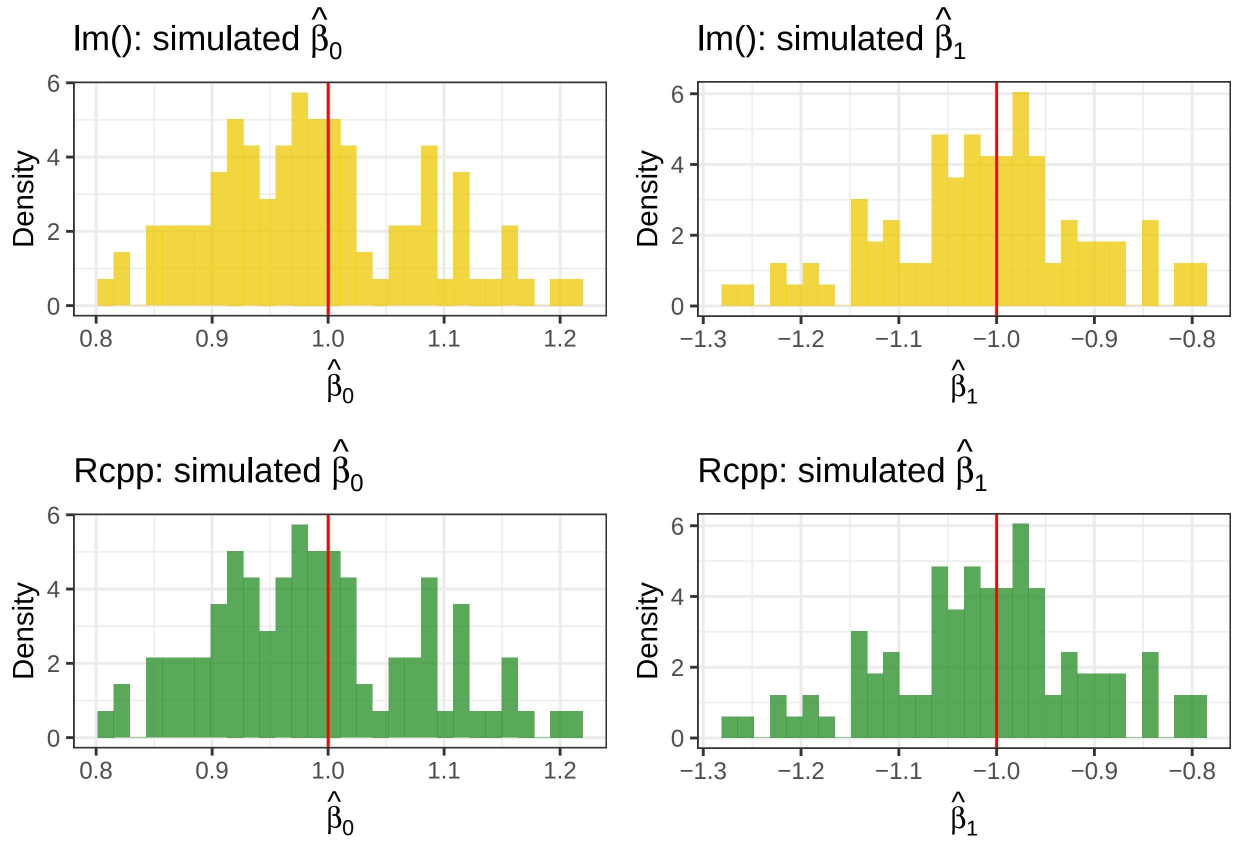

Figure 1: Histogram for the estimated regression coeefficient

Table 1: Performace for both methods: lm() and and Rcpp

	lm() function	Rcpp
Computational Time (secs)	0.00066 (SD = 0.00102)	0.00021 (SD = 0.00022)
Bias: b0	-0.01041 (SD = 0.0902)	-0.01041 (SD = 0.0902)
Bias: b1	-0.0112 (SD = 0.1023)	-0.0112 (SD = 0.1023)
Coverage Probability: b0	0.98 (SD = 0.1407)	0.98 (SD = 0.1407)
Coverage Probability: b1	0.95 (SD = 0.219)	0.95 (SD = 0.219)
MSE: b0	0.0082 (SD = 0.0096)	0.0082 (SD = 0.0096)
MSE: b1	0.0105 (SD = 0.0152)	0.0105 (SD = 0.0152)
Predictive MSE	0.9709 (SD = 0.1516)	0.9709 (SD = 0.1516)

According to Figure 1, we notice that the histograms of the simulated $\hat{\beta}_0$ for both `lm()` and `SimpLinR()` functions are the same. This is also true for the simulated $\hat{\beta}_1$. Table 1 shows the performance of the model in terms of computational time, bias, and mean squared error.

According to the results, we might notice that both `lm()` and Rcpp produce the same results in terms of the bias of the estimates, the coverage probability, the mean squared error for the regression coefficient, and the predicted mean squared error. The computation time, however, is the only aspect that makes these two methods appear different. Based on the results, we might see that Rcpp works faster than `lm()` by around 3.143 times.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE)
library(foreach)
library(doParallel)
library(tidyverse)
library(ggplot2)
library(latex2exp)
library(gridExtra)
library(knitr)
library(kableExtra)
library(SimpLin)
### User-define functions
meanSD <- function(x, dplace = 4){
  mm <- round(mean(x), digits = dplace)
  ss <- round(sd(x), digits = dplace)
  paste0(mm, " (SD = ", ss, ")")
}

### Simulate the data
set.seed(213, kind = "L'Ecuyer-CMRG")
registerDoParallel(5)
simDat <- foreach(t = 1:100) %dopar% {

  x <- rnorm(100)
  e <- rnorm(100)
  y <- 1 - x + e

  cbind(x, y)

}
stopImplicitCluster()

### Save the simulated data
saveRDS(simDat, "/Users/kevinkvp/Desktop/Github Repo/SimpLin/data/simulated.RData")
rm(simDat)
simDat <- readRDS("/Users/kevinkvp/Desktop/Github Repo/SimpLin/data/simulated.RData")

### Base R
set.seed(321, kind = "L'Ecuyer-CMRG")
registerDoParallel(5)
resultBase <- foreach(t = 1:100) %dopar% {

  start_time <- Sys.time()
  modBase <- lm(simDat[[t]][, "y"] ~ simDat[[t]][, "x"])
  timeBase <- difftime(Sys.time(), start_time) ### Run Time
  coefBase <- modBase$coefficients ### Coefficients
  predBase <- as.matrix(modBase$fitted.values) ### Predicted Value
  confBase <- confint(modBase) ### Confidence Interval
  list(time = timeBase, coef = coefBase, conf = confBase, pred = predBase)

}
stopImplicitCluster()
```

```r
### Rcpp
set.seed(132, kind = "L'Ecuyer-CMRG")
registerDoParallel(5)
resultRcpp <- foreach(t = 1:100) %dopar% {

  start_time <- Sys.time()
  modRcpp <- SimpLinR(simDat[[t]][, "x"], simDat[[t]][, "y"])
  timeRcpp <- difftime(Sys.time(), start_time) ### Run Time
  coefRcpp <- t(modRcpp$estimates) ### Coefficients
  predRcpp <- modRcpp$predicted_val ### Predicted Value
  confRcpp <- modRcpp$conf_int ### Confidence Interval
  list(time = timeRcpp, coef = coefRcpp, conf = confRcpp, pred = predRcpp)

}
stopImplicitCluster()

### Retrieve the estimated regression coefficients
coefBase <- t(sapply(1:100, function(x){resultBase[[x]]$coef}))
colnames(coefBase) <- c("V1",  "V2")
coefRcpp <- t(sapply(1:100, function(x){resultRcpp[[x]]$coef}))

### Histogram
b0Base <- ggplot(data.frame(coefBase), aes(x = V1)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30,
                 fill = "gold2", alpha = 0.75) +
  geom_vline(xintercept = 1, col = "red") +
  theme_bw() +
  labs(x = TeX("$\\hat{\\beta}_{0}$"),
       y = "Density",
       title = TeX("lm(): simulated $\\hat{\\beta}_{0}$"))

b1Base <- ggplot(data.frame(coefBase), aes(x = V2)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30,
                 fill = "gold2", alpha = 0.75) +
  geom_vline(xintercept = -1, col = "red") +
  theme_bw() +
  labs(x = TeX("$\\hat{\\beta}_{1}$"),
       y = "Density",
       title = TeX("lm(): simulated $\\hat{\\beta}_{1}$"))

b0Rcpp <- ggplot(as.data.frame(coefRcpp), aes(x = V1)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30,
                 fill = "forestgreen", alpha = 0.75) +
  geom_vline(xintercept = 1, col = "red") +
  theme_bw() +
  labs(x = TeX("$\\hat{\\beta}_{0}$"),
       y = "Density",
       title = TeX("Rcpp: simulated $\\hat{\\beta}_{0}$"))

b1Rcpp <- ggplot(as.data.frame(coefRcpp), aes(x = V2)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30,
                 fill = "forestgreen", alpha = 0.75) +
  geom_vline(xintercept = -1, col = "red") +
  theme_bw() +
  labs(x = TeX("$\\hat{\\beta}_{1}$"),
       y = "Density",
       title = TeX("Rcpp: simulated $\\hat{\\beta}_{1}$"))

grid.arrange(b0Base, b1Base, b0Rcpp, b1Rcpp)

trueCoef <- matrix(c(1, -1), nrow = 100, ncol = 2, byrow = T)

### Run Time
timeBase <- sapply(1:100, function(x){resultBase[[x]]$time}) %>% meanSD(5)
timeRcpp <- sapply(1:100, function(x){resultRcpp[[x]]$time}) %>% meanSD(5)

### Bias
biasBase <- apply(coefBase - trueCoef, 2, meanSD, 5)
biasRcpp <- apply(coefRcpp - trueCoef, 2, meanSD, 5)

### Coverage
covBase <- sapply(1:100, function(x){c((resultBase[[x]]$conf[1, 1] <= 1) & (1 <= resultBase[[x]]$conf[1
                             (resultBase[[x]]$conf[2, 1] <= -1) & (-1 <= resultBase[[x]]$conf[2, 2]))}) 
  apply(1, meanSD)

covRcpp <- sapply(1:100, function(x){c((resultRcpp[[x]]$conf[1, 1] <= 1) & (1 <= resultRcpp[[x]]$conf[1
                             (resultRcpp[[x]]$conf[2, 1] <= -1) & (-1 <= resultRcpp[[x]]$conf[2, 2]))}) 
  apply(1, meanSD)

### MSE of the predictor
msepBase <- apply(((coefBase - trueCoef)^2), 2, meanSD)
msepRcpp <- apply(((coefRcpp - trueCoef)^2), 2, meanSD)

### MSE for the prediction
mserBase <- sapply(1:100, function(x){mean((resultBase[[x]]$pred - simDat[[x]][, "y"])^2)}) %>%
  meanSD()

mserRcpp <- sapply(1:100, function(x){mean((resultRcpp[[x]]$pred - simDat[[x]][, "y"])^2)}) %>%
  meanSD()

### Create the report table
rbind(c(timeBase, timeRcpp), c(biasBase[1], biasRcpp[1]),
      c(biasBase[2], biasRcpp[2]), c(covBase[1], covRcpp[1]),
      c(covBase[2], covRcpp[2]), c(msepBase[1], msepRcpp[1]),
      c(msepBase[2], msepRcpp[2]), c(mserBase, mserRcpp)) %>%
  data.frame() %>%
  `rownames<-`(c("Computational Time (secs)", "Bias: b0", "Bias: b1",
                 "Coverage Probability: b0", "Coverage Probability: b1",
                 "MSE: b0", "MSE: b1", "Predictive MSE")) %>%
  kable(col.names = c("lm() function", "Rcpp"),
        caption = "Performace for both methods: lm() and and Rcpp")
```