STAT 600: Homework 1

Hyungjoon Kim, Colorado State University

January 30, 2024

Problem 1, 2

I made an `R` package named `SimpLin`, and uploaded to my own `github` repository. You may check the structure and code of the package **this link**. You can also install that package including the vignette using the following code:

```
devtools::install_github("hjkim8987/STAT600-2024S/HW1/SimpLin",
                         build_vignettes = TRUE)
```

Problem 3

I compared the performance of `lm` with `SimpLinR` using `rbenchmark` library. Each simulation data has $n = 100$ observations, and the number of simulations is also 100. I used 7 cores for the parallelization, and `SimpLinR` is almost twice faster than the built-in `lm` function. Even though the result of `lm` function contains much more information than `SimpLinR`, using `Rcpp` makes a huge improvement of speed.

```
library(SimpLin)

library(foreach)
library(doParallel)
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(rbenchmark)

n_iter <- 100
n_obs <- 100

n_cores <- detectCores()

registerDoParallel(n_cores - 1)

benchmark(
  "SimpLinR" = {
    set.seed(600)

    ans <- foreach(i = 1:n_iter, .combine = "c") %dopar% {
      x <- rnorm(n_obs)
      eps <- rnorm(n_obs)
      y <- 1 - x + eps
```

```r
      return(as.list(SimpLinR(x, y)))
    }
  },
  "lm" = {
    set.seed(600)

    ans <- foreach(i = 1:n_iter, .combine = "c") %dopar% {
      x <- rnorm(n_obs)
      eps <- rnorm(n_obs)
      y <- 1 - x + eps

      return(list(lm(y ~ x)))
    }
  },

  replications = 100,
  columns = c("test", "replications", "elapsed",
              "relative", "user.self", "sys.self")
)
```

```
##       test replications elapsed relative user.self sys.self
## 2       lm          100   5.069      2.1     1.069    1.924
## 1 SimpLinR          100   2.414      1.0     0.747    1.210
```

```r
stopImplicitCluster()
```

Problem 4

I attached three tables which summarize the simulation. According to the simulation, the built-in `lm` function and `SimpLin` give the exactly same results. They both have the same estimates, confidence intervals, bias, and so on. However, despite the simulation was implemented using the same-size data, `lm` took much more time for the fit.

```r
n_iter <- 100
n_obs <- 100

set.seed(600)

elapsed_time <- matrix(0, nrow = n_iter, ncol = 2)
bias <- matrix(0, nrow = n_iter, ncol = 4)
coverage <- matrix(0, nrow = n_iter, ncol = 4)
mse <- matrix(0, nrow = n_iter, ncol = 2)
pmse <- matrix(0, nrow = n_iter, ncol = 2)
beta_est <- matrix(0, nrow = n_iter, ncol = 4)

true_beta <- c(1, -1)
degrees_of_freedom <- n_obs - length(true_beta)
for(i in 1:n_iter){
  x <- rnorm(n_obs)
  eps <- rnorm(n_obs)
  y_true <- true_beta[1] + true_beta[2] * x
  y <- y_true + eps

  t1 <- Sys.time()
```

```r
  fit_lm <- lm(y ~ x)
  t2 <- Sys.time()
  fit_rcpp <- SimpLinR(x, y)
  t3 <- Sys.time()

  elapsed_time[i, ] <- c(t2 - t1, t3 - t2)
  bias[i, ] <- c(true_beta - fit_lm$coefficients,
                 true_beta - c(fit_rcpp$coefficients))
  coverage[i, ] <- c((confint(fit_lm)[1, 1] < true_beta[1]) *
                       (confint(fit_lm)[1, 2] > true_beta[1]),
                     (confint(fit_lm)[2, 1] < true_beta[2]) *
                       (confint(fit_lm)[2, 2] > true_beta[2]),
                     (fit_rcpp$confidence.intervals[1, 1] < true_beta[1]) *
                       (fit_rcpp$confidence.intervals[1, 2] > true_beta[1]),
                     (fit_rcpp$confidence.intervals[2, 1] < true_beta[2]) *
                       (fit_rcpp$confidence.intervals[2, 2] > true_beta[2]))
  mse[i, ] <- c((t(fit_lm$residuals) %*% fit_lm$residuals)[1, 1] /
                  degrees_of_freedom,
                (t(fit_rcpp$residuals) %*% fit_rcpp$residuals)[1, 1] /
                  degrees_of_freedom)
  pmse[i, ] <- c((t(y_true - fit_lm$fitted.values) %*%
                    (y_true - fit_lm$fitted.values))[1, 1] / n_obs,
                 (t(y_true - fit_rcpp$fitted.values) %*%
                    (y_true - fit_rcpp$fitted.values))[1, 1] / n_obs)
}
```

Table 1: Average Elapsed Time, MSE, and PMSE

	Elapsed Time	MSE	PMSE
`lm`	0.000259	1.001616	0.019266
`SimpLin`	0.000012	1.001616	0.019266

Table 2: Average Coverage Rate of 95% Confidence Interval

	β_0	β_1
`lm`	0.97	0.97
`SimpLin`	0.97	0.97

Table 3: Average Bias of the Estimation

	β_0	β_1
`lm`	0.0143687	-0.00069
`SimpLin`	0.0143687	-0.00069