Class 3 Problem Set

POS6933: Computational Social Science

Truscott (Spring 2026)

Note: Students should always aim to produce publication-worthy tables and figures. Unless otherwise stated, tables should be rendered using stargazer::(), while figures can be rendered using ggplot2::() or plot(). Regardless, tables and figures should always be presented with necessary formatting – e.g., (sub)title, axis (variable) labels and titles, a clearly-identifiable legend and key, etc. Problem sets must always be compiled using LaTex or RMarkdown and include the full coding routine (with notes explaining your implementation) used to complete each problem (10pts).

---

1. Write a function to sample at least 100 draws 100k times from a distribution of your choice before performing a secondary task to recover the mean value. Afterwards, compare the performance of your task given a serial versus parallel approach with at least three different sizes of cores being allocated. For this comparison, construct a table using `stargazer(html)` to record the computation strategy, number of cores, completion time, mean value, and 95 percent confidence intervals. (5pts)

```
recover_values_normal <- function(number_values, mean, sd){

  temp_values <- rnorm(number_values, mean = mean, sd = 5) # Recover Values
  return(mean(temp_values)) # Recover Mean

} # Recover Values

set.seed(1234) # Random Seed
seeds <- sample(1:100000, size = 100, replace = F)

serial_means <- data.frame()
serial_start <- Sys.time()
for (i in 1:100){
  set.seed(seeds[i]) # Set Random Seed
  temp_mean <- recover_values_normal(number_values = 100000, mean = 10, sd = 5) # Temp Run
  serial_means <- c(serial_means, temp_mean) # Append to Existing Vector
}
serial_end <- Sys.time()
serial_time <- round((serial_end - serial_start)[[1]], 3) # Seconds

{
  serial_mean <- mean(unlist(serial_means))
  serial_se   <- sd(unlist(serial_means)) / sqrt(length(unlist(serial_means)))
  serial_lower <- serial_mean - 1.96 * serial_se
  serial_upper <- serial_mean + 1.96 * serial_se
```

```r
} # Serial Mean & CIs

parallel_means <- list() # List to Store Means by Core Allocation
cores <- c(2, 4, 6) # Different Allocation Sizes

for (i in cores) {

  recover_values_normal <- function(number_values, mean, sd){

  temp_values <- rnorm(number_values, mean = mean, sd = sd) # Recover Values
  return(mean(temp_values)) # Recover Mean

}

  cl <- makeCluster(i)  # Create a socket cluster
  doParallel::registerDoParallel(cl) # Activate

  clusterExport(cl, c('recover_values_normal', 'seeds'))  # Allocate Global Environment Objects & Funct

  parallel_means_vector <- c()
  parallel_start <- Sys.time() # System Start

  parallel_run <- parLapply(cl, 1:100, function(i) {
  set.seed(seeds[i])  # set a different seed for each iteration
  recover_values_normal(number_values = 100000, mean = 10, sd = 5)
}) # Run 100 Times

  parallel_end <- Sys.time()
  parallel_time <- round((parallel_end - parallel_start)[[1]], 3)

  stopCluster(cl) # Stop Cluster

  parallel_means[[as.character(i)]] <- list(
    time = parallel_time,
    means = unlist(parallel_run)
  )


} # Run Parallel



summary_serial_parallel <- data.frame(
  method = c('Serial', rep('Parallel', 3)),
  cores_allocated = c(1, cores),
  time_completion = c(serial_time, sapply(c("2", "4", "6"), function(i) parallel_means[[i]]$time)),
  mean = c(serial_mean, sapply(c("2", "4", "6"), function(i) mean(parallel_means[[i]]$means))),
  lower_ci = c(serial_lower,  sapply(c("2", "4", "6"), function(i) {
    x <- parallel_means[[i]]$means
    m <- mean(x)
    se <- sd(x) / sqrt(length(x))
    lower <- m - 1.96 * se
    return(lower)})),
```

```
  upper_ci = c(serial_upper,
               sapply(c("2", "4", "6"), function(i) {
    x <- parallel_means[[i]]$means
    m <- mean(x)
    se <- sd(x) / sqrt(length(x))
    upper <- m + 1.96 * se
    return(upper)}))
) %>%
  mutate(mean = round(mean, 2),
         lower_ci = round(lower_ci, 3),
         upper_ci = round(upper_ci, 3)) %>%
  rename(
    Method = method,
    `Cores Allocated` = cores_allocated,
    `Completion Time` = time_completion,
    Mean = mean,
    `Lower CI` = lower_ci,
    `Upper CI` = upper_ci
  ) %>%
  as_tibble()

stargazer::stargazer(tibble::as.tibble(summary_serial_parallel),
                     summary = F,
                     type = 'text') # Produce Table
```

```
##
## ======================================================================
##    Method  Cores Allocated Completion Time Mean Lower CI Upper CI
## ----------------------------------------------------------------------
## 1  Serial         1             0.484       10   9.998    10.004
## 2 Parallel        2             0.296       10   9.998    10.004
## 3 Parallel        4             0.216       10   9.998    10.004
## 4 Parallel        6             0.162       10   9.998    10.004
## ----------------------------------------------------------------------
```

2. Create an artificial dataset consisting of one dependent variable and at least five independent variables (6 total). The dataset should contain no fewer than 100,000 observations. For replication purposes, label the dependent variable as y and the independent variables as x1 through x10. You may specify the distributions and variances of these variables at your discretion, though y should be a linear combination of atleast three predictors plus random error. Afterwards, create a function that randomly samples 10,000 observations without replacement from this dataset, recovers coefficients from a linear model, and subsequently estimates predictive values (or probabilities) using `predict()` across an expanded grid (`expand.grid()`) that ranges the scope of values for x1-x10. Completing this process 1,000 times using a serial configuration before transitioning and repeating using parallel with at least half + 1 cores available on your personal computer. For this comparison, construct a table using `stargazer(html)` to record the computation strategy, number of cores, and completion time. (5pts)

```
n = 100000
x1 <- rnorm(n, mean = 0, sd = 1)
x2 <- rnorm(n, mean = 5, sd = 2)
x3 <- runif(n, min = -1, max = 1)
x4 <- rnorm(n, mean = 10, sd = 5)
x5 <- rexp(n, rate = 1/2)
```

```r
y <- 3 + 1.5*x1 - 2*x2 + 0.5*x3 + rnorm(n, mean = 0, sd = 5) # Y is Linear Combination of Predictors +

sim_data <- data.frame(y, x1, x2, x3, x4, x5)

summary(sim_data)
```

```
##        y                 x1                  x2                x3
##  Min.   :-36.323   Min.   :-4.377543   Min.   :-3.601   Min.   :-0.999988
##  1st Qu.:-11.413   1st Qu.:-0.676814   1st Qu.: 3.651   1st Qu.:-0.502137
##  Median : -6.980   Median :-0.004536   Median : 5.002   Median :-0.002153
##  Mean   : -6.988   Mean   :-0.002866   Mean   : 5.002   Mean   :-0.001452
##  3rd Qu.: -2.530   3rd Qu.: 0.670748   3rd Qu.: 6.349   3rd Qu.: 0.496817
##  Max.   : 22.435   Max.   : 4.415982   Max.   :14.189   Max.   : 0.999975
##        x4                x5
##  Min.   :-13.164   Min.   : 0.000092
##  1st Qu.:  6.627   1st Qu.: 0.564625
##  Median : 10.012   Median : 1.380849
##  Mean   : 10.011   Mean   : 1.995345
##  3rd Qu.: 13.373   3rd Qu.: 2.781522
##  Max.   : 32.045   Max.   :21.354077
```

```r
set.seed(1234) # Random Seeds
seeds <- sample(1:100000, size = 1000, replace = F)

########################################
# Serial
########################################
serial_completion <- system.time(
  for (i in 1:1000){
  set.seed(seeds[i]) # Replication
  sample_data <- sim_data %>%
    slice_sample(n = 10000)
  temp_model <- lm(y ~ x1 + x2 + x3 + x4 + x5, data = sample_data) #
  newdata <- expand.grid(
    x1 = seq(min(sim_data$x1),max(sim_data$x1), sd(sim_data$x1)*2),
    x2 = seq(min(sim_data$x2),max(sim_data$x2), sd(sim_data$x2)*2),
    x3 = seq(min(sim_data$x3),max(sim_data$x3), sd(sim_data$x3)*2),
    x4 = seq(min(sim_data$x4),max(sim_data$x4), sd(sim_data$x4)*2),
    x5 = seq(min(sim_data$x5),max(sim_data$x5), sd(sim_data$x5)*2)
  )
  predictions <- predict(temp_model, newdata)
})[['elapsed']] # Run 100 Times, Recording Completion Time Once Complete


########################################
# Parallel
########################################

cores <- c(2, 4, 6)
parallel_completion_times <- c()
for (i in cores){

  parallel_completion <- system.time({
```

```
  cl <- makeCluster(i)
  doParallel::registerDoParallel(cl)
  clusterExport(cl, c('seeds', 'sim_data'))
  clusterEvalQ(cl, {
  library(dplyr)
})
  parallel_run <- parLapply(cl, 1:1000, function(i) {
    set.seed(seeds[i])
    sample_data <- sim_data %>% slice_sample(n = 10000)
    temp_model <- lm(y ~ x1 + x2 + x3 + x4 + x5, data = sample_data)
    newdata <- expand.grid(
      x1 = seq(min(sim_data$x1), max(sim_data$x1), sd(sim_data$x1) * 2),
      x2 = seq(min(sim_data$x2), max(sim_data$x2), sd(sim_data$x2) * 2),
      x3 = seq(min(sim_data$x3), max(sim_data$x3), sd(sim_data$x3) * 2),
      x4 = seq(min(sim_data$x4), max(sim_data$x4), sd(sim_data$x4) * 2),
      x5 = seq(min(sim_data$x5), max(sim_data$x5), sd(sim_data$x5) * 2)
    )
    predict(temp_model, newdata = newdata)
  })

  stopCluster(cl)
})[['elapsed']]

  parallel_completion_times <- c(parallel_completion_times, parallel_completion)


} # Run 100 Times, Recording Completion Time Once Complete Each Core Allocation

########################################
# Combine and Produce Table
########################################

summary_serial_parallel <- data.frame(
  `Method` = c('Serial', rep('Parallel', 3)),
  `Cores` = c(1, cores),
  `Completion Time` = round(c(serial_completion, parallel_completion_times), 3)
) %>%
  as_tibble()

stargazer::stargazer(summary_serial_parallel, summary = F, type = 'text')
```

```
##
## ================================
##    Method  Cores Completion.Time
## --------------------------------
## 1  Serial    1        9.73
## 2 Parallel   2        16.71
## 3 Parallel   4        15.34
## 4 Parallel   6        14.55
## --------------------------------
```