# pset 2 Labor

```r
#========================================#
# ==== Load packages and clear data ====
#========================================#

library(data.table)
library(Matrix)
library(lmtest)
library(sandwich)
library(broom)
library(ggplot2)
library(stats)
# clear objects and script
rm(list = ls(pos = ".GlobalEnv"), pos = ".GlobalEnv")
options(scipen = 999)
cat("\f")

#==============================#
# ==== Question 2  part 4 ====
#==============================#

  #=================================#
  # ==== geneerate random data ====
  #=================================#

    # set n_col and n_row
    n_col <- 10
    n_row <- 100
    n_cell <- n_col*n_row

    # create random matrices
    y_data <- matrix(runif(n_row, 0, 100), nrow = n_row, ncol = 1)
    x_data <- matrix(runif(n_cell, 0, 1), nrow = n_row, ncol = n_col)

  #=================================#
  # ==== write function for q2 ====
  #=================================#

    # commented out, but usefull for line by line debug
    # x = x_data
    # y = y_data

    # function
    mat_reg <- function(x = NULL, y = NULL, opt_chol = FALSE, CI_level = .95){

      # get matrix size parameters
      n_col <- ncol(x)
      n_row <- nrow(x)

    #========================#
    # ==== estimate beta ====
#========================#

  # check which inverse function to use
  if(!opt_chol){

    # use standard inverse
    B <- Matrix::solve(Matrix::crossprod(x, x))%*%(Matrix::crossprod(x, y))
  }else{

    # use cholesy inverse
    chol_m <- chol(Matrix::crossprod(x, x))
    B<- chol2inv(chol_m)%*%(Matrix::crossprod(x, y))

  }

#=====================#
# ==== estimate V ====
#=====================#

  # calculate residuals
  my_resid <- y - x%*%B

  # calculate middle part of variance matrix. the mear
  M_diag <- diag(as.numeric(my_resid^2*(n_row/(n_row-n_col))), nrow = n_row, ncol = n_row)
  M <- (t(x) %*% M_diag %*% x)


  # see if I need to use cholesky
  if(!opt_chol){

    # calculate asymptotic variance
    V <- solve(crossprod(x, x)) %*% M  %*% solve(crossprod(x, x))

  }else{

    A_inv <-   chol2inv(chol_m) %*% M  %*% chol2inv(chol_m)
    V <- A_inv

    }
  sqrt(diag(V))

#======================#
# ==== other stats ====
#======================#

  # start by putting beta and  diagonal of variance in a data.table
  out_dt <- data.table(beta = as.numeric(B), V_hat = diag(V) )

  # calculate standard errors
  out_dt[, se := sqrt(V_hat)]

  # calculate t test
  out_dt[, t_test := beta/(se)]
```

```
      # calculate p values
      out_dt[, p_value := 2*(1- pt((abs(t_test)), n_row - n_col))]

      # calculate confidence interval
      out_dt[, CI_L := beta - (se) * qt(1-((1-CI_level)/2), n_row )]
      out_dt[, CI_U := beta + (se) * qt(1-((1-CI_level)/2), n_row )]

      # drop v_hat cause I dont need it
      out_dt[, V_hat := NULL]

      # create list to return
      out_list <- list()

      out_list[["results"]] <- out_dt
      out_list[["varcov"]] <- V

      return(out_list)

}

#=======================================#
# ==== run function on random data ====
#=======================================#

  # run on random data with and without cholesky
  reg_1 <- mat_reg(x = x_data, y = y_data, opt_chol = FALSE)
  reg_2 <- mat_reg(x = x_data, y = y_data, opt_chol = TRUE)

  # compare coefficients, differences are just floating point errors
  coeff_diff <- reg_1[["results"]][, beta] - reg_2[["results"]][, beta]

  # compare varcov NOTE: differences are just floating point errors
  all.equal(reg_1$varcov, reg_2$varcov)
  reg_1$varcov -  reg_2$varcov

#============================#
# ==== Question 2 part 5 ====
#============================#

  #===========================#
  # ==== matrix function ====
  #===========================#

    # load daata #note paste is so it fits on pdf in markdown
    lalonde_dt <- fread(pasate0("C:/Users/Nmath_000/Documents/MI_school/Second",
                                "Year/675 Applied Econometrics/hw/hw1/LaLonde_1986.csv"))

    # grab y matrix
    y_la <- as.matrix(lalonde_dt[, earn78])

    # create other vars for regression
    lalonde_dt[, educ_sq := educ^2]
    lalonde_dt[, black_earn74 := black*earn74]
    lalonde_dt[, const := 1]

     # grab x vars
    x_vars <- c("treat", "black", "age", "educ",
                "educ_sq", "earn74","black_earn74",
                "u74","u75")

    # make x matrix
    x_la <- as.matrix(lalonde_dt[, c("const", x_vars), with = FALSE])

    # run function on this data
    lalonde_reg <- mat_reg(x = x_la, y = y_la)

    # grab the results
    results_2_5_a <- lalonde_reg[["results"]]

    # add in coef label
    results_2_5_a[, variable := c("const", x_vars)]

    # put variables in front
    setcolorder(results_2_5_a, c("variable", setdiff( colnames(results_2_5_a), "variable")))

  #====================#
  # ==== using lm ====
  #====================#

    # get regression formula
    reg_form <- as.formula(paste("earn78~", paste(x_vars, collapse="+")))

    # run regression
    lalonde_lm <- lm(reg_form, lalonde_dt)

    # get summary, NOTE: these are NOT robust standard errors
    lalong_lm_dt <- summary(lalonde_lm)$coefficients

    # get robust standard errors. I use HC2 to match my math above
    # any differnces are floating point errors
    lm_robust <- coeftest(lalonde_lm, vcov = vcovHC(lalonde_lm, type="HC1"))

    results_2_5_b <- data.table(tidy(lm_robust))

#======================#
# ==== Question 3 ====
#======================#


  #==================#
  # ==== neyman ====
  #==================#

    # 3.1.a calculate ATE
    TDM <- lalonde_dt[treat == 1, mean(earn78)] - lalonde_dt[treat == 0, mean(earn78)]
    # get variance for treatment and no treatnment
    s1_sq <- lalonde_dt[treat == 1, var(earn78)]
    s0_sq <- lalonde_dt[treat == 0, var(earn78)]

    # get V_tdm
    V_tdm <- s1_sq/lalonde_dt[treat == 1, .N] + s0_sq/lalonde_dt[treat == 0, .N]

    # get standard error
    se_tdm <- sqrt(V_tdm)

    # constuct 95% convidence interval
     tdm_CI_L <- TDM - se_tdm * qnorm(.975)
     tdm_CI_U <- TDM + se_tdm * qnorm(.975)

     # put together resuts
     results_3_1_b <- data.table("TDM est" = TDM,
                                 "Conservative SE" = se_tdm,
                                 "CI Lower" = tdm_CI_L,
                                 "CI Upper" =  tdm_CI_U)

  #=================#
  # ==== fisher ====
  #=================#

     # definitions for line by line debug
     # in_data= lalonde_dt
     # y_var = "earn78"
     # treat_var = "treat"
     # opt_test_stat= "DM"
     # n_iter = 10
     # null_hyp = 5000

    # write function for fisher p value
    fisher_p <- function(in_data       = NULL,
                         y_var         = NULL,
                         treat_var     = NULL,
                         null_hyp      = 0,
                         opt_test_stat = "DM",
                         n_iter        = 1999){

      # check that a test has ben species
      if(!opt_test_stat %chin% c("DM", "KS")){
        stop("Specify either DM ot KS test")
      }

      # check for non-zero null under the KS test (function doesn't do that)
      if(opt_test_stat == "KS" & null_hyp != 0){
        stop("The KS test is not compatibe with a non-zero null at the moment")
      }

      # copy data so I can create y(0) and y(1) cols without altering input data set
      data_c <- copy(in_data)
```

```r
  # create colums for sharp null treated and untreated y variables
  data_c[get(treat_var) == 1, y_1 := get(y_var) ]
  data_c[get(treat_var) == 0, y_1 := get(y_var) + null_hyp ]
  data_c[get(treat_var) == 0, y_0 := get(y_var) ]
  data_c[get(treat_var) == 1, y_0 := get(y_var) - null_hyp ]

  # create a data.table for the results of bootstrap
  sim_data <- data.table(iteration = c(1:(n_iter+1)))

  # get the number of treated vars
  n_treat <- nrow(data_c[get(treat_var) == 1, ])
  n_row <- nrow(data_c)

  # do actual test
  if(opt_test_stat == "DM"){

    # get mean of treatment
    m_t <- data_c[get(treat_var) == 1, mean(get(y_var))]

    # get mean of untreated
    m_unt <- data_c[get(treat_var) == 0, mean(get(y_var))]

    test_1 <-m_t - m_unt - null_hyp

  }
  if(opt_test_stat == "KS"){
    ksout <- suppressWarnings(ks.test(data_c[get(treat_var) == 1, get(y_var)],
                                      data_c[get(treat_var) == 0, get(y_var)] ))
    test_1 <- ksout$statistic
  }

  # put results of actual data in table
  sim_data[iteration == 1, test := test_1]

  # for each iteration
  for(i in 2:(n_iter + 1)){

    # create a permutation
    sample_i_1 <- sample.int(n = n_row, size = n_treat)
    sample_i_0 <- setdiff(c(1: n_row), sample_i_1)

    # calculate the averate treatment effect for this given sample
    if(opt_test_stat == "DM"){

      test_i <- data_c[sample_i_1, mean(y_1)] - data_c[sample_i_0, mean(y_0)] - null_hyp
    }
    if(opt_test_stat == "KS"){
      ksout <- suppressWarnings(ks.test(data_c[sample_i_1, y_1], data_c[sample_i_0, y_0] ))
      test_i <- ksout$statistic
    }


    # store this value in the data table
```

```r
      sim_data[ i, test := test_i]
    }

    # get absolute value and rank of the tests
    sim_data[, abs_test := abs(test)]
    sim_data[, test_rank := frank(abs_test)]

    # get p value
    p_value <- (nrow(sim_data) - sim_data[iteration == 1, test_rank] + 1)/nrow(sim_data)

    return(p_value)


  }

  # run function on data
  results_3_2_a_DM <- fisher_p(in_data       = lalonde_dt,
                               y_var         = "earn78",
                               treat_var     = "treat",
                               null_hyp      = 0,
                               opt_test_stat = "DM",
                               n_iter        = 999)

  results_3_2_a_KS <- fisher_p(in_data       = lalonde_dt,
                               y_var         = "earn78",
                               treat_var     = "treat",
                               null_hyp      = 0,
                               opt_test_stat = "KS",
                               n_iter        = 999)

  # make it fancy for output
  results_3_2_a_DM <- data.table("DM P value" =   results_3_2_a_DM )
  results_3_2_a_KS <- data.table("KS P value" =   results_3_2_a_KS )
  #=============================================#
  # ==== construct 95% confidence interval ====
  #=============================================#

    # run fcuntions on a range of data
    grid <- seq(5000,-1500,-5)

    dm_p_list <- lapply(grid,
                        fisher_p,
                        in_data= lalonde_dt,
                        y_var = "earn78",
                        treat_var = "treat",
                        opt_test_stat= "DM",
                        n_iter = 999)

    results_3_2_b <- data.table(hyp_treat = grid, p_value = dm_p_list)

    # make it pretty
    setnames(results_3_2_b, "hyp_treat", "Hypothesized Treatment Effect")
```

```
#==============================#
# ==== Power calculations ====
#==============================#

# plot attributes from EA
plot_attributes <- theme(plot.background = element_rect(fill = "lightgrey"),
                        panel.grid.major.x = element_line(color = "gray90"),
                        panel.grid.minor  = element_blank(),
                        panel.background = element_rect(fill = "white",
                                                        colour = "black") ,
                        panel.grid.major.y = element_line(color = "gray90"),
                        text = element_text(size= 30),
                        plot.title = element_text(vjust=0,
                                                  colour = "#0B6357",
                                                  face = "bold",
                                                  size = 30))


  # write power function
  power_function <- function(x, se= NULL) {
    1 - pnorm(qnorm(0.975)-x/se) + pnorm(-qnorm(0.975)-x/se)
  }

  # plot function
  power_plot <- ggplot(data = data.frame(x = 0), mapping = aes(x = x))
 power_plot <- power_plot + stat_function(fun = power_function,
                                         args = list(se=results_3_1_b$`Conservative SE`),
                                         color = "blue")
 power_plot <- power_plot + xlim(-5000,5000) + xlab("tau") + ylab("Power")  + plot_attributes
 power_plot

#=========================#
# ==== find needed n ====
#=========================#

 # Parameterize the equation
 p     = 2/3
 tau   = 1000

 # Write down the power function, which implicitly defines N
 Fun <- function(N, s.0 = s0_sq, s.1 = s1_sq){
   -0.8 + 1 - pnorm(qnorm(0.975)-tau/sqrt(1/N*s.1*(1/p)+1/N*s.0*(1/(1-p)))) +
     pnorm(-qnorm(0.975)-tau/sqrt(1/N*s.1*(1/p)+1/N*s.0*(1/(1-p))))
 }

 # Solve for N
 N.sol <- uniroot(Fun,c(0,100000000))$root


#=====================#
# ==== save stuff ====
#=====================#
```

```r
# save plot
png( paste0("C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_1_tex/",
             "power_func_r.png", height = 800, width = 800, type = "cairo"))
print(power_plot)
dev.off()

# save results #badcode so lazy
res_objects <- ls()[grepl("results", ls())]

save_tex_tables <- function(obj_name = NULL){

  table <- get(obj_name)

  print(xtable(table, type = "latex"),
        file = paste0("C:/Users/Nmath_000/Documents/Code/courses/econ 675/PS_1_tex/",
                      obj_name, ".tex"),
        include.rownames = FALSE,
        floating = FALSE)

}

lapply(res_objects, save_tex_tables)
```