Appendix: Source code

2024-04-21

```r
knitr::opts_chunk$set(echo = TRUE)
rm(list = ls())
gc()
```

```
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 475831 25.5    1027411 54.9   664211 35.5
## Vcells 891650  6.9    8388608 64.0  1814572 13.9
```

```r
set.seed(1)
options(digits=6)
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```r
pacman::p_load(
  tidyverse,
  mlr3verse,
  rpart.plot,
  ranger,
  future,
  mlr3pipelines,
  igraph,
  glmnet
)
```

Data

```r
df <- read_csv("./input/MROZ_mini.csv")
```

```
## Rows: 428 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): educ, fatheduc, lwage
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
head(df, 30)
```

```
## # A tibble: 30 x 3
##     educ fatheduc  lwage
##    <dbl>    <dbl>  <dbl>
##  1    12        7 1.21
##  2    12        7 0.329
##  3    12        7 1.51
##  4    12        7 0.0921
##  5    14       14 1.52
##  6    12        7 1.56
##  7    16        7 2.12
##  8    12        3 2.06
##  9    12        7 0.754
## 10    12        7 1.54
## # i 20 more rows
```

Question 1-1

```r
##### Question 1 #####

const <- rep(1, nrow(df))
Mat_X <- as.matrix(cbind(const, df[, 1]))
Mat_y <- as.matrix(df[, 3])

# Computing beta_hat
numerical_beta <- solve(t(Mat_X) %*% Mat_X) %*% (t(Mat_X) %*% Mat_y)

### Q1 Answer ###
print(paste("beta_0: ", numerical_beta[1]))
```

```
## [1] "beta_0:  -0.185196923871551"
```

```r
print(paste("beta_1: ", numerical_beta[2]))
```

```
## [1] "beta_1:  0.108648664436512"
```

Question 1-2

```r
##### Question 2 #####

# Definition of the objective function
compute_ols <- function(theta, df) {
  beta_0 <- theta[1]
  beta_1 <- theta[2]
  J <- 0.0
  for (i in 1:nrow(df)) {
    add_J <- (df$lwage[[i]] - beta_0 - (beta_1*df$educ[[i]]))** 2
    J <- J + add_J
```

```r
  }
  return(J)
}

# Set initial value of theta
initial_theta <- c(0, 0)

# Minimizing the objective function by using optim function
result <- optim(par = initial_theta, fn = compute_ols, df = df, method = "BFGS")
result
```

```
## $par
## [1] -0.184886  0.108633
##
## $value
## [1] 197.001
##
## $counts
## function gradient
##       23        6
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
### Q2 Answer ###
print(paste("Numerical beta_0: ", numerical_beta[1]))
```

```
## [1] "Numerical beta_0:  -0.185196923871551"
```

```r
print(paste("Analytical beta_0: ", result$par[1]))
```

```
## [1] "Analytical beta_0:  -0.184885798843542"
```

```r
print(paste("Numerical beta_1: ", numerical_beta[2]))
```

```
## [1] "Numerical beta_1:  0.108648664436512"
```

```r
print(paste("Analytical beta_1: ", result$par[2]))
```

```
## [1] "Analytical beta_1:  0.108632651079137"
```

Question 1-3

```r
# compute Asymptotic SE of OLS estimator

# Hayashi p.123 calculating sample mean of S
compute_S_hat <- function(theta, df) {
  S_hat <- matrix(0, ncol = 2, nrow = 2)
  for (i in 1:nrow(df)) {
    x_i_mat <- Mat_X[i, ]
    epsilon_hat <- as.numeric(df$lwage[[i]] - t(x_i_mat) %*% theta)
    add_S_hat <- (epsilon_hat ^ 2) * (x_i_mat %*% t(x_i_mat))
    S_hat <- S_hat + add_S_hat
  }
  S_hat <- (1/nrow(df)) * S_hat
  return(S_hat)
}

S_xx <- (1/nrow(df)) * (t(Mat_X) %*% Mat_X)
S_hat <- compute_S_hat(numerical_beta, df)

# Computing the asymptotic variance estimator
Avar_est <- solve(S_xx) %*% S_hat %*% solve(S_xx)

# Computing the asymptotic SE for beta_0 and beta_1
Asy_std_beta_0 <- sqrt ((1/nrow(df)) * Avar_est[1])
Asy_std_beta_1 <- sqrt ((1/nrow(df)) * Avar_est[4])

### Q3 Answer ###
print(paste("Asymptotic standard error beta 0: ", Asy_std_beta_0))
```

```
## [1] "Asymptotic standard error beta 0:  0.170348665439351"
```

```r
print(paste("Asymptotic standard error beta 1: ", Asy_std_beta_1))
```

```
## [1] "Asymptotic standard error beta 1:  0.0133839371539942"
```

Question 1-6

```r
##### Question 6 #####

# Define Z as IV
Mat_Z <- as.matrix(cbind(const, df[, 2])) # const + futheduc

# get IV estimator
P_Z = Mat_Z %*% solve(t(Mat_Z) %*% Mat_Z) %*% t(Mat_Z)
numerical_beta_IV <- solve(t(Mat_X) %*% P_Z %*% Mat_X) %*% (t(Mat_X) %*% P_Z %*% Mat_y)
numerical_beta_IV
```

```
##           lwage
## const 0.4411035
## educ  0.0591735
```

```r
# Compute asymptotic SE of IV estimator based on Hansen p. 354

# Compute epsilon_hat
compute_epsilon_hat <- function(theta, df) {
  epsilon_hat <- 0
  for (i in 1:nrow(df)) {
    x_i_mat <- Mat_X[i, ]
    z_i_mat <- Mat_Z[i, ]
    add_epsilon_hat <- as.numeric(df$lwage[[i]] - t(x_i_mat) %*% theta)
    add_epsilon_hat <- (add_epsilon_hat) ^ 2
    epsilon_hat <- epsilon_hat + add_epsilon_hat
  }
  epsilon_hat <- (1/nrow(df)) * epsilon_hat
  return(epsilon_hat)
}

Q_xz <- (1/nrow(df)) * (t(Mat_X) %*% Mat_Z)
Q_zx <- (1/nrow(df)) * (t(Mat_Z) %*% Mat_X)
Q_zz <- (1/nrow(df)) * (t(Mat_Z) %*% Mat_Z)
epsilon_hat <- compute_epsilon_hat(numerical_beta_IV, df)

# Computing the asymptotic variance estimator
edge_comp <- solve(Q_xz %*% solve(Q_zz) %*% Q_zx)
Avar_est_IV <- edge_comp * epsilon_hat

# Computing the asymptotic SE for beta_0 and beta_1
Asy_std_beta_IV_0 <- sqrt ((1/nrow(df)) * Avar_est_IV[1])
Asy_std_beta_IV_1 <- sqrt ((1/nrow(df)) * Avar_est_IV[4])

### Q6 Answer ###

# print beta IV
print(paste("Numerical beta_IV_0: ", numerical_beta_IV[1]))
```

```
## [1] "Numerical beta_IV_0:  0.441103500025292"
```

```r
print(paste("Numerical beta_IV_1: ", numerical_beta_IV[2]))
```

```
## [1] "Numerical beta_IV_1:  0.0591734740659255"
```

```r
# Print Asy SE for beta
print(paste("Asymptotic standard error beta IV 0: ", Asy_std_beta_IV_0))
```

```
## [1] "Asymptotic standard error beta IV 0:  0.44505826449563"
```

```r
print(paste("Asymptotic standard error beta IV 1: ", Asy_std_beta_IV_1))
```

```
## [1] "Asymptotic standard error beta IV 1:  0.0350595718842378"
```

Question 2-3