Appendix: Source code

2024-04-21

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

```
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 475915 25.5    1027651 54.9   664211 35.5
## Vcells 892251  6.9    8388608 64.0  1814572 13.9
```

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

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

```
pacman::p_load(
  tidyverse,
  fixest
)
```

Data

```
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.
```

```
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

```
##### 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"
```

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

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

Question 1-2

```
##### 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
  }
  return(J)
}

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

```r
# 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)
```

```r
    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, ]
```

```r
    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"
```

Kinoko Takenoko Data

```r
kntk_df <- read_csv("./input/data_KinokoTakenoko.csv")
```

```
## New names:
## Rows: 1110 Columns: 4
```

```
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): occasion dbl (3): ...1, id, choice
## 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.
## * `` -> `...1`
```

```
# Set price
kntk_df <- kntk_df %>%
    mutate(p_kino = if_else(occasion == "X1", 200, 0),
           p_kino = if_else(occasion == "X2", 170, p_kino),
           p_kino = if_else(occasion == "X3", 240, p_kino),
           p_kino = if_else(occasion == "X4", 200, p_kino),
           p_kino = if_else(occasion == "X5", 200, p_kino),
           p_take = if_else(occasion == "X1", 200, 0),
           p_take = if_else(occasion == "X2", 200, p_take),
           p_take = if_else(occasion == "X3", 200, p_take),
           p_take = if_else(occasion == "X4", 250, p_take),
           p_take = if_else(occasion == "X5", 180, p_take)
           )

head(kntk_df, 30)
```

```
## # A tibble: 30 x 6
##     ...1    id occasion choice p_kino p_take
##    <dbl> <dbl> <chr>     <dbl>  <dbl>  <dbl>
##  1     1     1 X1            2    200    200
##  2     2     2 X1            2    200    200
##  3     3     3 X1            0    200    200
##  4     4     4 X1            0    200    200
##  5     5     5 X1            0    200    200
##  6     6     6 X1            0    200    200
##  7     7     7 X1            0    200    200
##  8     8     8 X1            2    200    200
##  9     9     9 X1            1    200    200
## 10    10    10 X1            2    200    200
## # i 20 more rows
```

Question 2-4

```
##### Question 2-4 #####
compute_loglikelihood <- function(theta, df) {
  alpha_kino <- theta[1]
  alpha_take <- theta[2]
  beta <- theta[3]

  # Calculate probability for i, j, k
  df <- df %>%
    mutate(denominator = 1 + exp(alpha_kino - beta*p_kino) + exp(alpha_take - beta*p_take),
           prob = if_else(choice == 1, exp(alpha_kino - beta*p_kino) / denominator, 0),
           prob = if_else(choice == 2, exp(alpha_take - beta*p_take) / denominator, prob),
```

```
          prob = if_else(choice == 0, 1 / denominator, prob))

  N = max(df$id)
  lf <- 0.0
  for (i in 1:N) {
    for (k in 1:5) {
      # get P_{ijk}
      prob <- df %>%
        filter(id == i & occasion == paste0("X", k)) %>%
        pull(prob)

      add_lf <- log(prob)

      lf <- lf + add_lf
    }
  }
  return(lf)
}

# OLS To get the initial value
reg_df <- kntk_df %>%
  mutate(y = if_else(choice == 1, 1, 0),
         x = if_else(choice == 1, p_kino, 0),
         x = if_else(choice == 2, p_take, x))
model <- feols(y ~  1   + x,
               reg_df, vcov="White"
)
etable(model)
```

```
##                                 model
## Dependent Var.:                     y
##
## Constant         0.0197*** (0.0035)
## x               0.0019*** (9.59e-5)
## _______________ ___________________
## S.E. type       Heteroskedast.-rob.
## Observations                  1,110
## R2                          0.17220
## Adj. R2                     0.17145
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Maximizing the Log-Likelihood by using optim function
initial_theta <- c(model$coefficients[1], model$coefficients[1], model$coefficients[2])
start.time <- Sys.time()
MLE_res <- optim(par = initial_theta, fn = compute_loglikelihood, df = kntk_df, method='BFGS', control =
end.time <- Sys.time()
end.time - start.time
```

```
## Time difference of 2.24403 mins
```

```r
#Answer Q2-4
MLE_res
```

```
## $par
## (Intercept) (Intercept)           x
##   7.2577831   7.8409494   0.0383363
##
## $value
## [1] -1074.95
##
## $counts
## function gradient
##       55       10
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
print(paste("alpha kinoko hat: ", MLE_res$par[1]))
```

```
## [1] "alpha kinoko hat:  7.2577830856353"
```

```r
print(paste("alpha takenoko hat: ", MLE_res$par[2]))
```

```
## [1] "alpha takenoko hat:  7.8409493751478"
```

```r
print(paste("beta: ", MLE_res$par[3]))
```

```
## [1] "beta:  0.0383362818089461"
```

```r
# Robustness check about initial value

for (i in seq(from = 0, to = 0.5, by = 0.5)) {
  initial_theta <- c(i, i, i)
  print(paste("Initial theta: ", c(i, i, i)))
  start.time <- Sys.time()
  MLE_res <- optim(par = initial_theta, fn = compute_loglikelihood, df = kntk_df, method='BFGS', control
  end.time <- Sys.time()
  end.time - start.time
  print(MLE_res)
}
```

```
## [1] "Initial theta:  0" "Initial theta:  0" "Initial theta:  0"
## $par
## [1] 7.2576237 7.8405483 0.0383394
##
## $value
## [1] -1074.95
```

```
##
## $counts
## function gradient
##       59       10
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## [1] "Initial theta:  0.5" "Initial theta:  0.5" "Initial theta:  0.5"
## $par
## [1] 7.3065392 7.8904109 0.0385854
##
## $value
## [1] -1074.94
##
## $counts
## function gradient
##      121       12
##
## $convergence
## [1] 0
##
## $message
## NULL
```