Assignment 2: Production Function Estimation

Kohei Kawaguchi

Simulate data

Consider the following production and investment process for $j = 1, \cdots, 1000$ firms across $t = 1, \cdots, 10$ periods.

The log production function is of the form:

$$y_{jt} = \beta_0 + \beta_l l_{jt} + \beta_k k_{jt} + \omega_{jt} + \eta_{jt},$$

where ω_{jt} is an anticipated shock and η_{jt} is an ex post shock.

The anticipated shocks evolve as:

$$\omega_{jt} = \alpha \omega_{j,t-1} + \nu_{jt},$$

where ν_{jt} is an i.i.d. normal random variable with mean 0 and standard deviation σ_ν. The ex post shock is an i.i.d. normal random variable with mean 0 and standard deviation σ_η.

The product price the same across firms and normalized at 1. The price is normalized at 1. The wage w_t is constant at 0.5.

Finally, the capital accumulate according to:

$$K_{j,t+1} = (1 - \delta) K_{jt} + I_{jt}.$$

We set the parameters as follows:

parameter	variable	value
β_0	`beta_0`	1
β_l	`beta_l`	0.2
β_k	`beta_k`	0.7
α	`alpha`	0.7
σ_η	`sigma_eta`	0.2
σ_ν	`sigma_nu`	0.5
σ_w	`sigma_w`	0.1
δ	`delta`	0.05

1. Define the parameter variables as above.

```
# size
J <- 1000
T <- 10

# production function
beta_0 <- 1
beta_l <- 0.2
beta_k <- 0.7
sigma_eta <- 0.2
```

```
# anticipated shocks
sigma_nu <- 0.5
alpha <- 0.7

# wage
sigma_w <- 0.1

# capital
delta <- 0.05
```

2. Write a function that returns the log output given l_{jt}, k_{jt}, ω_{jt}, and η_{jt} under the given parameter values according to the above production function and name it `log_production(l, k, omega, eta, beta_0, beta_l, beta_k)`.

```
# log production function
log_production <-
  function(l, k, omega, eta, beta_0, beta_l, beta_k) {
    y <- beta_0 + beta_l * l + beta_k * k + omega + eta
    return(y)
  }
```

Suppose that the labor is determined after ω_{jt} is observed, but before η_{jt} is observed, given the log capital level k_{jt}.

3. Derive the optimal log labor as a function of ω_{jt}, η_{jt}, k_{jt}, and wage. Write a function to return the optimal log labor given the variables and parameters and name it `log_labor_choice(k, wage, omega, beta_0, beta_l, beta_k, sigma_eta)`.

```
log_labor_choice <-
  function(k, wage, omega, beta_0, beta_l, beta_k, sigma_eta) {
    L <-
      ((1/wage) *
         exp(beta_0 + omega + sigma_eta^2/2) *
         beta_l *
         exp(k)^beta_k)^(1/(1 - beta_l))
    l <- log(L)
  return(l)
  }
```

As discussed in the class, if there is no additional variation in labor, the coefficient on the labor β_l is not identified. Thus, if we generate labor choice from the previous function, β_l will not be identified from the simulated data. To see this, we write a modified version of the previous function in which ω_{jt} is replaced with $\omega_{jt} + \iota_{jt}$, where ι_{jt} is an optimization error that follows an i.i.d. normal distribution with mean 0 and standard deviation 0.05. That is, the manager of the firm perceives as if the shock is $\omega_{jt} + \iota_{jt}$, even though the true shock is ω_{jt}.

4. Modify the previous function by including ι_{jt} as an additional input and name it `log_labor_choice_error(k, wage, omega, beta_0, beta_l, beta_k, iota, sigma_eta)`.

```
log_labor_choice_error <-
  function(k, wage, omega, beta_0, beta_l, beta_k, iota, sigma_eta) {
    L <-
      ((1/wage) *
         exp(beta_0 + omega + iota + sigma_eta^2/2) *
         beta_l *
         exp(k)^beta_k)^(1/(1 - beta_l))
    l <- log(L)
```

```
  return(l)
  }
```

Consider an investment process such that:

$$I_{jt} = (\delta + \gamma\omega_{jt})K_{jt},$$

where I_{jt} and K_{jt} are investment and capital in level. Set $\gamma = 0.1$, i.e., the investment is strictly increasing in ω_{jt}. The investment function should be derived by solving the dynamic problem of a firm. But here, we just specify it in a reduced-form.

5. Define variable γ and assign it the value. Write a function that returns the investment given K_{jt}, ω_{jt}, and parameter values, according to the previous equation, and name it `investment_choice(k, omega, gamma, delta)`.

```
gamma <- 0.1
investment_choice <-
  function(k, omega, gamma, delta) {
    I <- (delta + gamma * omega) * exp(k)
    return(I)
  }
```

Simulate the data first using the labor choice without optimization error and second using the labor choice with optimization error. To do so, we specify the initial values for the state variables k_{jt} and ω_{jt} as follows.

6. Draw k_{j1} from an i.i.d. normal distribution with mean 1 and standard deviation 0.5. Draw ω_{j1} from its stationary distribution (check the stationary distribution of AR(1) process). Draw a wage. Before simulating the rest of the data, set the seed at 1.

```
# optimization error
sigma_iota <- 0.05

# parameters for the initial state distribution
mean_k_initial <- 1
sigma_k_initial <- 0.5
sigma_v_initial <- sigma_nu / sqrt(1 - alpha^2)

# set seed
set.seed(1)

# draw initial state values
wage <- 0.5
df <- expand.grid(j = 1:J, t = 1) %>%
  tibble::as_tibble() %>%
  dplyr::mutate(
    k = rnorm(J, mean_k_initial, sigma_k_initial),
    omega = rnorm(J, 0, sigma_v_initial),
    wage = wage
  )
df
```

```
## # A tibble: 1,000 x 5
##        j     t     k   omega  wage
##    <int> <dbl> <dbl>   <dbl> <dbl>
##  1     1     1 0.687  0.795    0.5
##  2     2     1 1.09   0.779    0.5
##  3     3     1 0.582 -0.610    0.5
```

```
##  4     4     1 1.80   0.148    0.5
##  5     5     1 1.16   0.0486   0.5
##  6     6     1 0.590 -1.16     0.5
##  7     7     1 1.24   0.568    0.5
##  8     8     1 1.37  -1.34     0.5
##  9     9     1 1.29  -0.873    0.5
## 10    10     1 0.847  0.699    0.5
## # ... with 990 more rows
```

7. Draw optimization error ι_{jt} and compute the labor and investment choice of period 1. For labor choice, compute both types of labor choices.

```
# compute labor and investment
df <- df %>%
  dplyr::mutate(iota = rnorm(J, 0, sigma_iota)) %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    l = log_labor_choice(k, wage, omega, beta_0, beta_l, beta_k, sigma_eta),
    l_error = log_labor_choice_error(k, wage, omega, beta_0, beta_l, beta_k, iota, sigma_eta),
    I = investment_choice(k, omega, gamma, delta)
  ) %>%
  dplyr::ungroup()
df
```

```
## # A tibble: 1,000 x 9
##        j     t     k   omega  wage     iota      l l_error       I
##    <int> <dbl> <dbl>   <dbl> <dbl>    <dbl>  <dbl>   <dbl>   <dbl>
##  1     1     1 0.687  0.795    0.5 -0.0443   1.72   1.67    0.257
##  2     2     1 1.09   0.779    0.5 -0.0961   2.06   1.94    0.381
##  3     3     1 0.582 -0.610    0.5  0.0810  -0.123 -0.0218 -0.0196
##  4     4     1 1.80   0.148    0.5  0.0260   1.89   1.92    0.391
##  5     5     1 1.16   0.0486   0.5 -0.00279  1.21   1.21    0.176
##  6     6     1 0.590 -1.16     0.5  0.0348  -0.809 -0.766  -0.120
##  7     7     1 1.24   0.568    0.5  0.00268  1.93   1.93    0.370
##  8     8     1 1.37  -1.34     0.5 -0.0655  -0.346 -0.428  -0.330
##  9     9     1 1.29  -0.873    0.5 -0.106    0.165  0.0327 -0.135
## 10    10     1 0.847  0.699    0.5 -0.0104   1.74   1.73    0.280
## # ... with 990 more rows
```

8. Draw ex post shock and compute the output according to the production function for both labor without optimization error and with optimization error. Name the output without optimization error `y` and the one with optimization error `y_error`.

```
# compute output
df <- df %>%
  dplyr::mutate(eta = rnorm(J, 0, sigma_eta)) %>%
  dplyr::rowwise() %>%
  dplyr::mutate(
    y = log_production(l, k, omega, eta, beta_0, beta_l, beta_k),
    y_error = log_production(l_error, k, omega, eta, beta_0, beta_l, beta_k)
  ) %>%
  dplyr::ungroup()
df
```

```
## # A tibble: 1,000 x 12
##        j     t     k   omega  wage     iota      l l_error       I     eta     y
##    <int> <dbl> <dbl>   <dbl> <dbl>    <dbl>  <dbl>   <dbl>   <dbl>   <dbl> <dbl>
```

```
##  1     1     1 0.687  0.795    0.5 -0.0443   1.72   1.67    0.257   0.148  2.77
##  2     2     1 1.09   0.779    0.5 -0.0961   2.06   1.94    0.381   0.0773 3.03
##  3     3     1 0.582 -0.610    0.5  0.0810  -0.123 -0.0218 -0.0196  0.259  1.03
##  4     4     1 1.80   0.148    0.5  0.0260   1.89   1.92    0.391  -0.161  2.62
##  5     5     1 1.16   0.0486   0.5 -0.00279  1.21   1.21    0.176  -0.321  1.79
##  6     6     1 0.590 -1.16     0.5  0.0348  -0.809 -0.766  -0.120   0.187  0.274
##  7     7     1 1.24   0.568    0.5  0.00268  1.93   1.93    0.370   0.361  3.19
##  8     8     1 1.37  -1.34     0.5 -0.0655  -0.346 -0.428  -0.330  -0.0113 0.539
##  9     9     1 1.29  -0.873    0.5 -0.106    0.165  0.0327 -0.135   0.377  1.44
## 10    10     1 0.847  0.699    0.5 -0.0104   1.74   1.73    0.280   0.316  2.96
## # ... with 990 more rows, and 1 more variable: y_error <dbl>
```

9. Repeat this procedure for $t = 1, \cdots 10$ by updating the capital and anticipated shocks, and name the resulting data frame `df_T`.

```
df_T <- df
for (t in 2:T) {
  # change time index
  df$t <- t

  # draw wage
  wage <- 0.5
  df$wage <- wage

  # update capital
  df <- df %>%
    dplyr::mutate(
      k = log((1 - delta) * exp(k) + I)
    )

  # update omega
  df <- df %>%
    dplyr::mutate(
      nu = rnorm(J, 0, sigma_nu),
      omega = alpha * omega + nu
    )

  # compute labor and investment
  df <- df %>%
    dplyr::mutate(iota = rnorm(J, 0, sigma_iota)) %>%
    dplyr::rowwise() %>%
    dplyr::mutate(
      l = log_labor_choice(k, wage, omega, beta_0, beta_l, beta_k, sigma_eta),
      l_error = log_labor_choice_error(k, wage, omega, beta_0, beta_l, beta_k, iota, sigma_eta),
      I = investment_choice(k, omega, gamma, delta)
    ) %>%
    dplyr::ungroup()
  # compute output
  df <- df %>%
    dplyr::mutate(eta = rnorm(J, 0, sigma_eta)) %>%
    dplyr::rowwise() %>%
    dplyr::mutate(
      y = log_production(l, k, omega, eta, beta_0, beta_l, beta_k),
      y_error = log_production(l_error, k, omega, eta, beta_0, beta_l, beta_k)
    ) %>%
```

```
    dplyr::ungroup()

  # append
  df_T <- dplyr::bind_rows(df_T, df)
}
df_T
```

```
## # A tibble: 10,000 x 13
##        j     t     k   omega  wage     iota      l l_error       I     eta     y
##    <int> <dbl> <dbl>   <dbl> <dbl>    <dbl>  <dbl>   <dbl>   <dbl>   <dbl> <dbl>
##  1     1     1 0.687  0.795    0.5 -0.0443   1.72   1.67    0.257   0.148  2.77
##  2     2     1 1.09   0.779    0.5 -0.0961   2.06   1.94    0.381   0.0773 3.03
##  3     3     1 0.582 -0.610    0.5  0.0810  -0.123 -0.0218 -0.0196  0.259  1.03
##  4     4     1 1.80   0.148    0.5  0.0260   1.89   1.92    0.391  -0.161  2.62
##  5     5     1 1.16   0.0486   0.5 -0.00279  1.21   1.21    0.176  -0.321  1.79
##  6     6     1 0.590 -1.16     0.5  0.0348  -0.809 -0.766  -0.120   0.187  0.274
##  7     7     1 1.24   0.568    0.5  0.00268  1.93   1.93    0.370   0.361  3.19
##  8     8     1 1.37  -1.34     0.5 -0.0655  -0.346 -0.428  -0.330  -0.0113 0.539
##  9     9     1 1.29  -0.873    0.5 -0.106    0.165  0.0327 -0.135   0.377  1.44
## 10    10     1 0.847  0.699    0.5 -0.0104   1.74   1.73    0.280   0.316  2.96
## # ... with 9,990 more rows, and 2 more variables: y_error <dbl>, nu <dbl>
```

```
save(df_T, file = "data/A2_df_T.RData")
```

10. Check the simulated data by making summary table.

```
df_T_summary <-
  foreach (i = 1:dim(df_T)[2], .combine = "rbind") %do% {
    df_T_i <- as.data.frame(df_T)[, i]
    row_i <- data.frame(
      N = length(df_T_i),
      Mean = mean(df_T_i, na.rm = TRUE),
      Sd = sd(df_T_i, na.rm = TRUE),
      Min = min(df_T_i, na.rm = TRUE),
      Max = max(df_T_i, na.rm = TRUE))
    rownames(row_i) <- colnames(df_T)[i]
    return(row_i)
  }
kable(df_T_summary)
```

	N	Mean	Sd	Min	Max
j	10000	500.5000000	288.6894251	1.0000000	1000.0000000
t	10000	5.5000000	2.8724249	1.0000000	10.0000000
k	10000	0.9797900	0.5838949	-1.2822534	3.2332312
omega	10000	-0.0055826	0.7025102	-2.5894171	2.6281307
wage	10000	0.5000000	0.0000000	0.5000000	0.5000000
iota	10000	-0.0000696	0.0502883	-0.1841453	0.1715419
l	10000	0.9799746	1.0965108	-3.3281023	4.9679634
l_error	10000	0.9798876	1.0971595	-3.3765433	4.9520674
I	10000	0.1793502	0.3006526	-1.2722627	3.2975332
eta	10000	0.0015825	0.2001539	-0.7650371	0.7455922
y	10000	1.8778479	1.1171035	-2.4680251	6.1228291
y_error	10000	1.8778305	1.1169266	-2.4777133	6.1196499
nu	10000	-0.0021155	0.4984324	-2.1513907	1.8253882

Estimate the parameters

For now, use the labor choice with optimization error.

1. First, simply regress y_{jt} on l_{jt} and k_{jt} using the least square method. This is likely to give an upwardly biased estimates on β_l and β_k. Why is it?

```
result_ols <-
  lm(data = df_T,
     formula = y_error ~ l_error + k)
summary(result_ols)
```

```
##
## Call:
## lm(formula = y_error ~ l_error + k, data = df_T)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.73002 -0.14117 -0.00071  0.13743  0.87983
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.892542   0.004058 219.966   <2e-16 ***
## l_error     0.997913   0.002396 416.454   <2e-16 ***
## k           0.007599   0.004503   1.688   0.0915 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2068 on 9997 degrees of freedom
## Multiple R-squared:  0.9657, Adjusted R-squared:  0.9657
## F-statistic: 1.408e+05 on 2 and 9997 DF,  p-value: < 2.2e-16
```

2. Second, take within-transformation on y_{jt}, l_{jt}, and k_{jt} and let Δy_{jt}, Δl_{jt}, and Δk_{jt} denote them. Then, regress Δy_{jt} on Δl_{jt}, and Δk_{jt} by the least squares method.

```
df_T_within <- df_T %>%
  dplyr::group_by(j) %>%
  dplyr::mutate(dy_error = y_error - mean(y_error),
                dl_error = l_error - mean(l_error),
                dk = k - mean(k)) %>%
  dplyr::ungroup()
result_within <-
  lm(data = df_T_within,
     formula = dy_error ~ - 1 + dl_error + dk)
summary(result_within)
```

```
##
## Call:
## lm(formula = dy_error ~ -1 + dl_error + dk, data = df_T_within)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72450 -0.13285 -0.00244  0.12931  0.77657
##
## Coefficients:
##            Estimate Std. Error t value Pr(>|t|)
## dl_error  0.9910916  0.0029548 335.413   <2e-16 ***
```

```
## dk          -0.0009029  0.0127539  -0.071    0.944
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1961 on 9998 degrees of freedom
## Multiple R-squared:  0.9184, Adjusted R-squared:  0.9184
## F-statistic: 5.629e+04 on 2 and 9998 DF,  p-value: < 2.2e-16
```

Next, we attempt to estimate the parameters using Olley-Pakes method. Estimate the first-step model of Olley-Pakes method:

$$y_{jt} = \beta_0 + \beta_1 l_{jt} + \phi(k_{jt}, I_{jt}) + \eta_{jt},$$

by approximating ϕ_t by a kernel function.

Remark that ϕ in general depends on observed and unobserved state variables. For this reason, in theory, ϕ should be estimated for each period. In this exercise, we assume ϕ is common across periods because we know that there is no unobserved state variables in the true data generating process. Moreover, we do not include w_t because we know that it is time -invariant. Do not forget to consider them in the actual data analysis.

You can use `npplreg` function of `np` package to estimate a partially linear model with a multivariate kernel. You first use `npplregbw` to obtain the optimal band width and then use `npplreg` to estimate the model under the optimal bandwidth. The computation of the optimal bandwidth is time consuming.

3. Return the summary of the first stage estimation and plot the fitted values against the data points.

```
result_1st_bw <-
  npplregbw(data = df_T,
            formula = y_error ~ l_error + k + I| k + I)
save(result_1st_bw, file = "data/A2_result_1st_bw.RData")
```

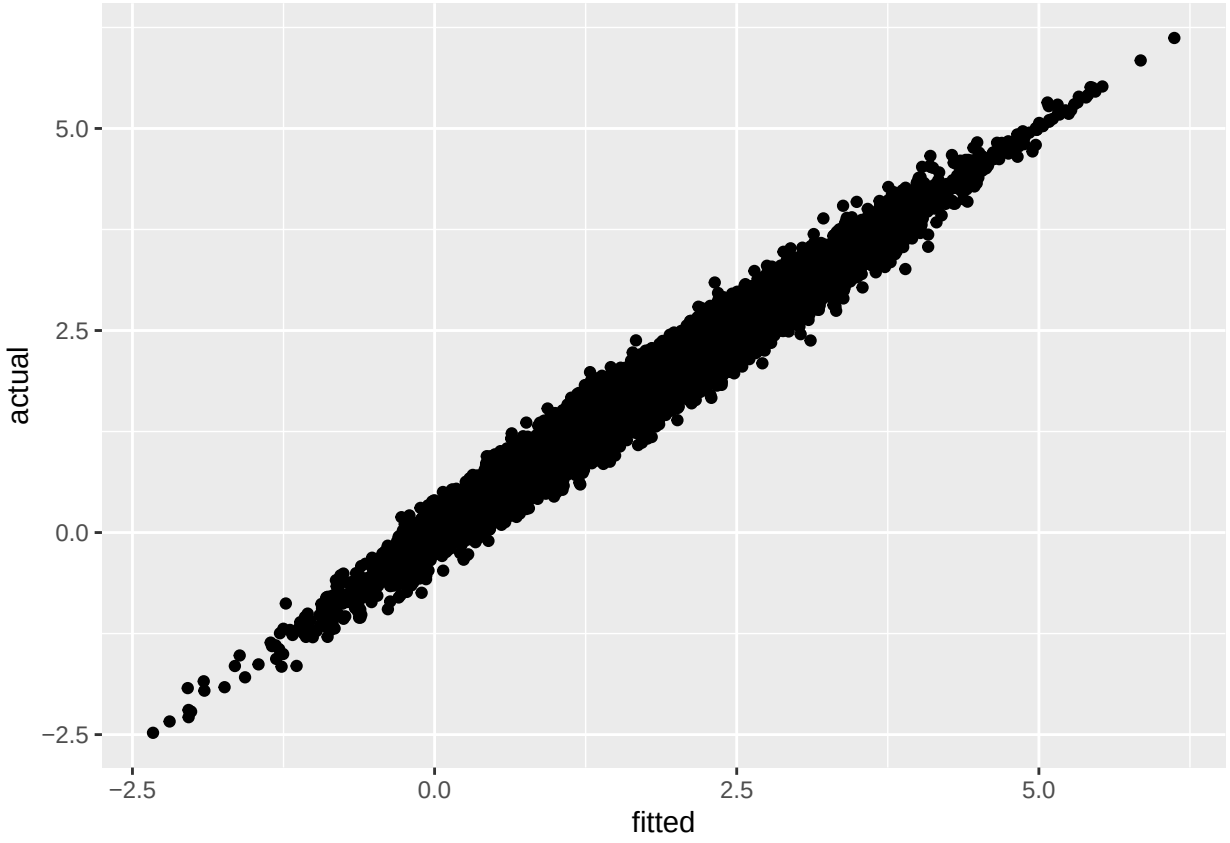
```
result_1st <-
  npplreg(data = df_T,
          formula = y_error ~ l_error + k + I| k + I,
          bws = result_1st_bw)
save(result_1st, file = "data/A2_result_1st.RData")
```

```
load(file = "data/A2_df_T.RData")
load(file = "data/A2_result_1st.RData")
summary(result_1st)
```

```
##
## Partially Linear Model
## Regression data: 10000 training points, in 5 variable(s)
## With 3 linear parametric regressor(s), 2 nonparametric regressor(s)
##
##                      y(z)
## Bandwidth(s): 0.07355058 0.01435558
##
##                      x(z)
## Bandwidth(s): 0.03908594 0.01191551
##               0.01397428 3.74952954
##               0.83529394 0.00398329
##
##                   l_error        k        I
## Coefficient(s): 0.2485295 2.355522 5.345144
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
```

```
##
## Residual standard error: 0.1934585
## R-squared: 0.970064
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 2
```

```
result_1st_plot <-
  data.frame(actual = df_T$y_error,
             fitted = fitted(result_1st))
ggplot(result_1st_plot, aes(x = fitted, y = actual)) +
  geom_point()
```



4. Check that β_l is not identified with the data without optimization error. Estimate the first stage model of Olley-Pakes with the labor choice without optimization error and report the result.

```
result_1st_unidentified_bw <-
  npplregbw(data = df_T,
            formula = y ~ l + k + I| k + I)
save(result_1st_unidentified_bw,
     file = "data/A2_result_1st_unidentified_bw.RData")
```

```
load(file = "data/A2_result_1st_unidentified_bw.RData")
result_1st_unidentified <-
  npplreg(data = df_T,
          formula = y ~ l + k + I| k + I,
          bws = result_1st_unidentified_bw)
save(result_1st_unidentified,
```

```
     file = "data/A2_result_1st_unidentified.RData")
```

```
load(file = "data/A2_result_1st_unidentified.RData")
summary(result_1st_unidentified)
```

```
##
## Partially Linear Model
## Regression data: 10000 training points, in 5 variable(s)
## With 3 linear parametric regressor(s), 2 nonparametric regressor(s)
##
##                      y(z)
## Bandwidth(s): 0.07347226 0.01437256
##
##                      x(z)
## Bandwidth(s): 0.02960021 0.009986945
##               0.01397428 3.749529542
##               0.83529394 0.003983290
##
##                         l        k         I
## Coefficient(s): 1.180628 2.034182 0.7805077
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
##
## Residual standard error: 0.1932285
## R-squared: 0.970116
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 2
```

Then, we estimate the second stage model of Olley-Pakes method:

$$y_{jt} - \hat{\beta}_l l_{jt} = \beta_0 + \beta_k k_{jt} + \alpha[\hat{\phi}(k_{j,t-1}, I_{j,t-1}) - \beta_0 - \beta_k k_{jt}] + \nu_{jt} + \eta_{jt}.$$

In this model, we do not have to non-parametetrically estimate the conditional expectation of ω_{jt} on $\omega_{j,t-1}$, because we know that the anticipated shock follows an AR(1) process. Remark that we in general have to non-parametrically estimate it.

The model is non-linear in parameters, because of the term $\alpha\beta_0$ and $\alpha\beta_k$. We estimate α, β_0, and β_k by a GMM estimator. The moment is:

$$g_{JT}(\alpha, \beta_0, \beta_k) \equiv \frac{1}{JT} \sum_{j=1}^{J} \sum_{t=1}^{T} \{y_{jt} - \hat{\beta}_l l_{jt} - \beta_0 - \beta_k k_{jt} - \alpha[\hat{\phi}(k_{j,t-1}, I_{j,t-1}) - \beta_0 - \beta_k k_{jt}]\} \begin{bmatrix} k_{jt} \\ k_{j,t-1} \\ I_{j,t-1} \end{bmatrix}.$$

5. Using the estimates in the first step, compute:

$$y_{jt} - \hat{\beta}_l l_{jt},$$

and:

$$\hat{\phi}(k_{j,t-1}, I_{j,t-1}),$$

for each j and t and save it as a data frame names `df_T_1st`.

```
df_T_1st <- df_T %>%
  dplyr::mutate(y_error_tilde =
                  y_error -
```

```
                  result_1st$xcoef["l_error"] * l_error) %>%
  dplyr::select(j, t, y_error_tilde)
phi <- fitted(result_1st) -
  result_1st$xcoef["l_error"] * df_T$l_error
df_T_1st$phi_t <- phi
df_T_1st <- df_T_1st %>%
  dplyr::arrange(j, t) %>%
  dplyr::group_by(j) %>%
  dplyr::mutate(phi_t_1 = dplyr::lag(phi_t, 1)) %>%
  dplyr::ungroup() %>%
  dplyr::select(-phi_t)
df_T_1st
```

```
## # A tibble: 10,000 x 4
##        j     t y_error_tilde phi_t_1[,1]
##    <int> <dbl>         <dbl>       <dbl>
##  1     1     1         2.34       NA
##  2     1     2         1.37        2.21
##  3     1     3         0.621       1.49
##  4     1     4         0.447       0.882
##  5     1     5         0.878       0.611
##  6     1     6         1.62        0.926
##  7     1     7         0.558       1.40
##  8     1     8         0.684       0.439
##  9     1     9         0.939       0.520
## 10     1    10         1.49        0.836
## # ... with 9,990 more rows
```

6. Compute a function that returns the value of $g_{JT}(\alpha, \beta_0, \beta_k)$ given parameter values, data, and `df_T_1st`, and name it `moment_OP_2nd`. Show the values of the moments evaluated at the true parameters.

```
moment_OP_2nd <-
  function(alpha, beta_0, beta_k, df_T, df_T_1st) {
    moment <- df_T %>%
      dplyr::group_by(j) %>%
      dplyr::mutate(
        k_t_1 = dplyr::lag(k, 1),
        I_t_1 = dplyr::lag(I, 1)
      ) %>%
      dplyr::ungroup() %>%
      dplyr::left_join(df_T_1st, by = c("j", "t")) %>%
      dplyr::filter(!is.na(phi_t_1)) %>%
      dplyr::mutate(
        g_jt = y_error_tilde - beta_0 - beta_k * k -
          alpha * (phi_t_1 - beta_0 - beta_k * k_t_1)
      )
    moment <-
      cbind(moment$g_jt * moment$k,
            moment$g_jt * moment$k_t_1,
            moment$g_jt * moment$I_t_1)
    moment <- apply(moment, 2, mean)
    return(moment)
  }
moment_OP_2nd(alpha, beta_0, beta_k, df_T, df_T_1st)
```

```
## [1] -0.018507303 -0.019038229 -0.003867714
```

Based on the moment, we can define the objective function of a generalized method of moments estimator with a weighting matrix W as:

$$Q_{JT}(\alpha, \beta_0, \beta_k) \equiv g_{JT}(\alpha, \beta_0, \beta_k)' W g_{JT}(\alpha, \beta_0, \beta_k).$$

7. Write a function that returns the value of $Q_{JT}(\alpha, \beta_0, \beta_k)$ given the vector of parameter values, data, and `df_T_1st`, and name it `objective_OP_2nd`. Setting W at the identity matrix, show the value of the objective function evaluated at the true parameters.

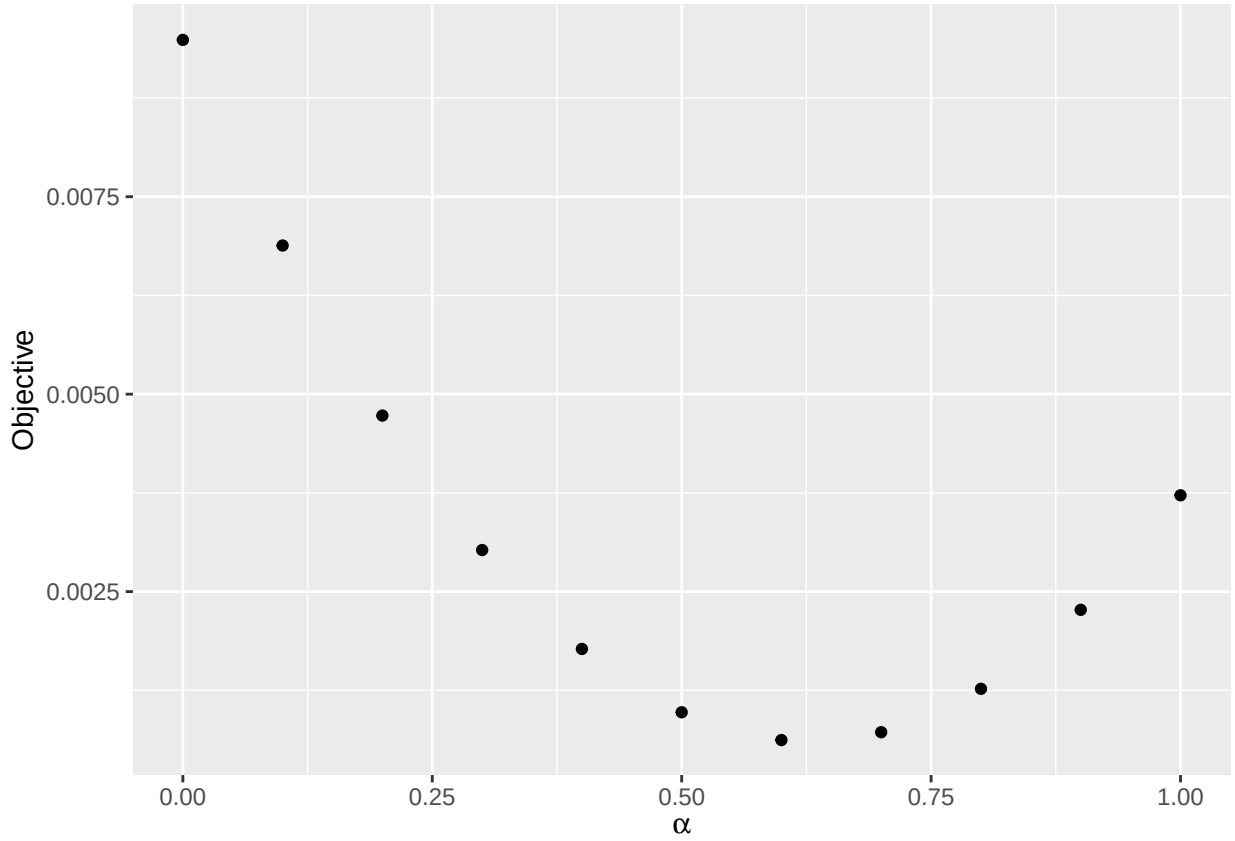
```
objective_OP_2nd <-
  function(theta, df_T, df_T_1st, W) {
    alpha <- theta[1]
    beta_0 <- theta[2]
    beta_k <- theta[3]
    moment <-
      moment_OP_2nd(alpha, beta_0, beta_k, df_T, df_T_1st)
    objective <- t(moment) %*% W %*% moment
    return(objective)
  }
W <- diag(3)
theta <- c(alpha, beta_0, beta_k)
objective_OP_2nd(theta, df_T, df_T_1st, W)
```

```
##              [,1]
## [1,] 0.0007199336
```

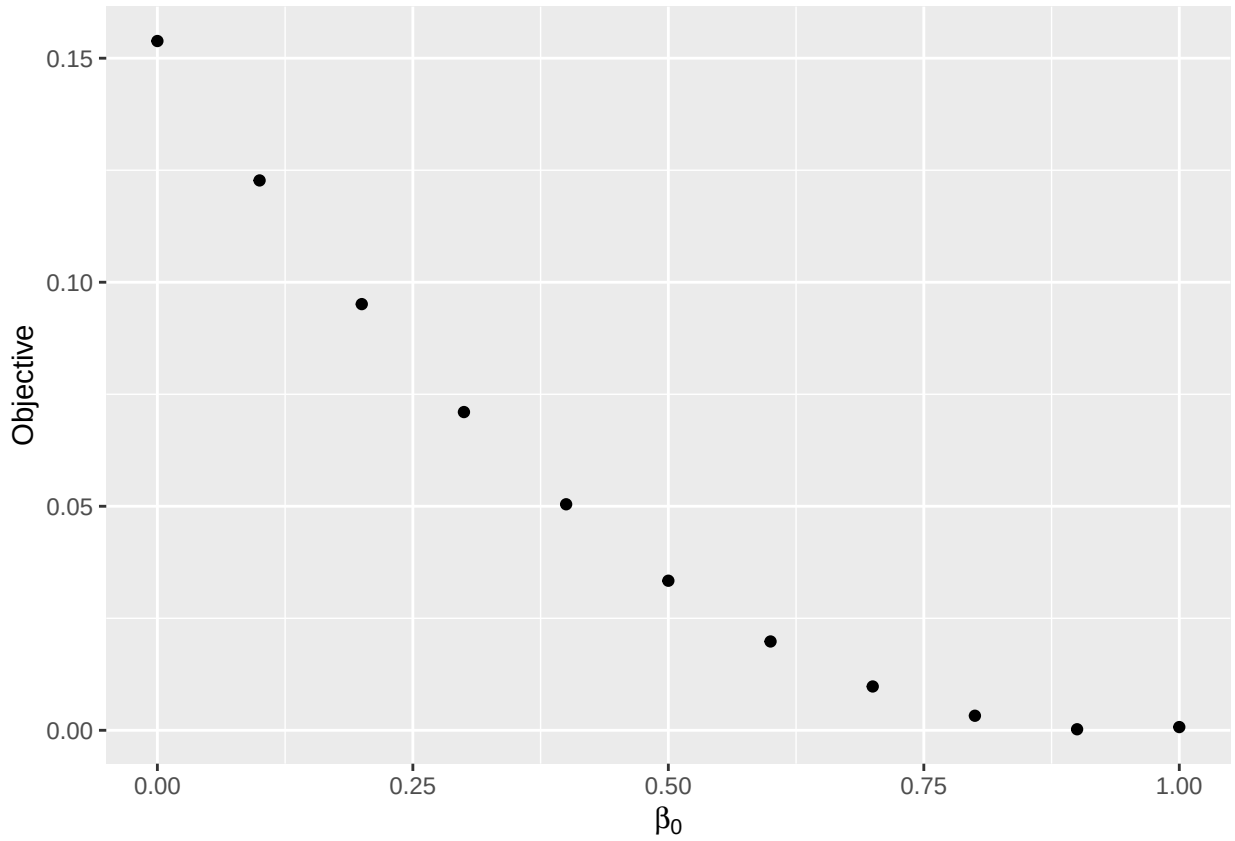
8. Draw the graph of the objective function when one of α, β_0, and β_k are changed from 0 to 1 by 0.1 while the others are set at the true value. Is the objective function minimized at around the true value?

```
objective_alpha <-
  foreach (i = seq(0, 1, 0.1),
           .combine = "rbind") %do% {
    objective_i <-
      objective_OP_2nd(c(i, beta_0, beta_k), df_T, df_T_1st, W)
    return(objective_i)
           }
objective_beta_0 <-
  foreach (i = seq(0, 1, 0.1),
           .combine = "rbind") %do% {
    objective_i <-
      objective_OP_2nd(c(alpha, i, beta_k), df_T, df_T_1st, W)
    return(objective_i)
           }
objective_beta_k <-
  foreach (i = seq(0, 1, 0.1),
           .combine = "rbind") %do% {
    objective_i <-
      objective_OP_2nd(c(alpha, beta_0, i), df_T, df_T_1st, W)
    return(objective_i)
           }
objective_plot <-
  data.frame(i = seq(0, 1, 0.1),
             alpha = objective_alpha,
             beta_0 = objective_beta_0,
             beta_k = objective_beta_k)
```

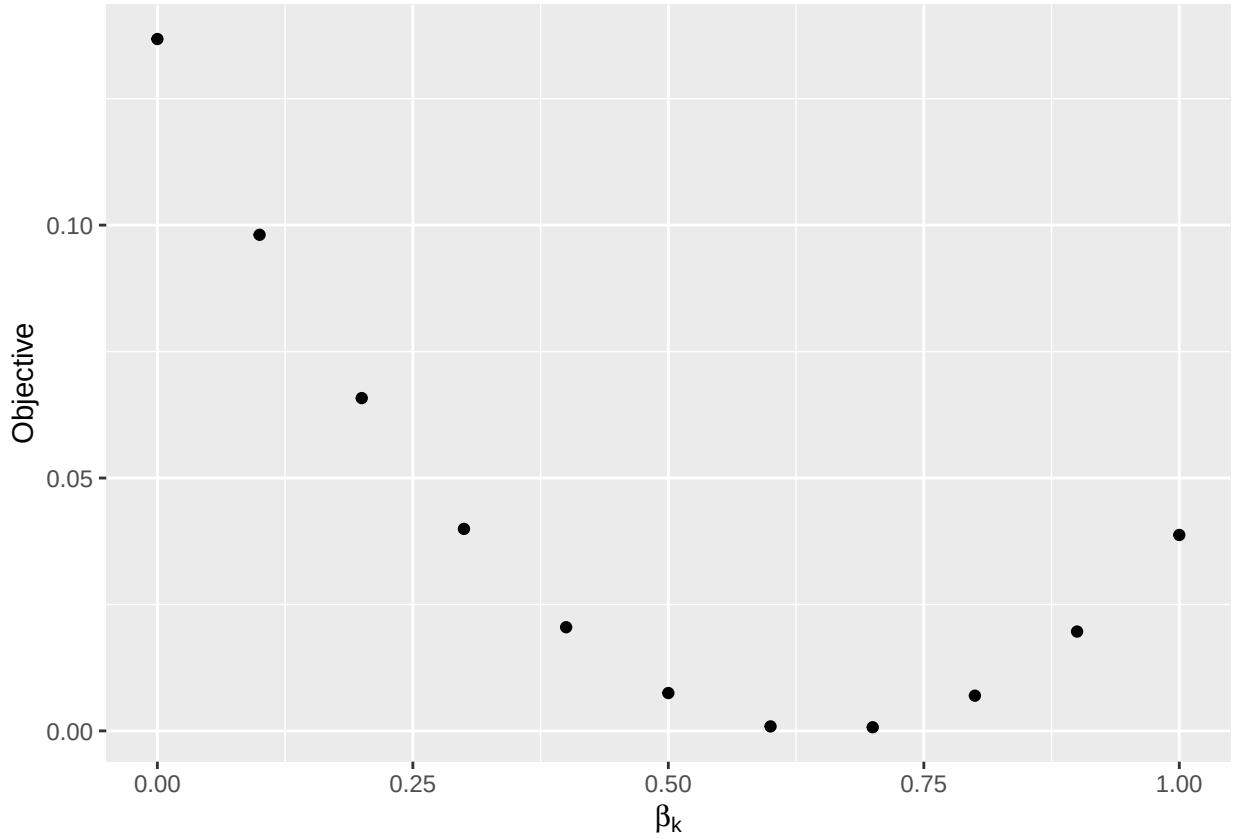
```
ggplot(data = objective_plot,
       aes(x = i, y = alpha)) +
  geom_point() + xlab(TeX("$\\alpha$")) + ylab("Objective")
```




```
ggplot(data = objective_plot,
       aes(x = i, y = beta_0)) +
  geom_point() + xlab(TeX("$\\beta_0$")) + ylab("Objective")
```




```
ggplot(data = objective_plot,
       aes(x = i, y = beta_k)) +
  geom_point() + xlab(TeX("$\\beta_k$")) + ylab("Objective")
```



9. Find the parameters that minimize the objective function using `optim`. You may use `L-BFGS-B` method to solve it.

```
theta <- c(alpha, beta_0, beta_k)
W <- diag(3)
result_2nd <-
  optim(par = theta,
        f = objective_OP_2nd,
        method = "L-BFGS-B",
        df_T = df_T,
        df_T_1st = df_T_1st,
        W = W)
result_2nd
```

```
## $par
## [1] 0.7020260 0.9766308 0.6693945
##
## $value
## [1] 1.994601e-07
##
## $counts
## function gradient
##       10       10
##
## $convergence
## [1] 0
##
```

```
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```