Exercise 1.2, Structural Econometrics

Giovanni Cavalcanti

2024-11-06

Problem 2: Random Coefficients Logit

Assume that $\sigma_1 \neq 0$ and $\epsilon_{ijm} \sim T1EV(1,0)$. Assume that prices are correlated with ξ_{jm}.

2.1 Derive the choice probabilities.

We can rewrite the utility function of individual i as:

$$U_{ijm} = V_{ijm} + \epsilon_{ijm} \quad ; \quad V_{ij} = (\delta_{jm} + \mu_{ijm} + \xi_j + \xi_{jm})$$

where $\delta_{jm} = -\alpha p_{jm} + \sum_{k=1}^{4} \beta_k x_{jmk} + \xi_{jm} \quad , \quad \mu_{ijm} = \sigma_1 v_i x_{jm4}$

μ_{ijm} captures correlation across choices, this correlation depends on characteristics t and unobserved taste shocks ν. So, at any market m, the probability of choosing j is given by

$$A_{ijm} = \{(\epsilon_i, \beta_i) : V_{ijm} + \epsilon_{im} \geq V_{ikm} + \epsilon_{ikm} \; \forall \; k \neq j\}$$
$$P(a_{im} = j) = P(V_{ijm} + \epsilon_{jm} \geq V_{ikm} + \epsilon_{km} \; \forall \; k \neq j)$$

Therefore, using the assumption that $\epsilon_i = (\epsilon_1, \ldots, \epsilon_{50})$ is i.i.d. across J, and the β_i vary over i with population density $f(\beta|\theta)$,

$$\begin{aligned} P(a_{im} = j) &= \int_{A_{ijm}} f((\epsilon_i, \beta_i))\, d(\epsilon_i, \beta_i) \\ &= \int_{\beta_i} f(\beta_i|\theta) \left(\int_{A_{ijm}} f(\epsilon_i)\, d\epsilon_i \right) d\beta_i \\ &= \int_{\beta_i} \frac{\exp(\beta_i' x_j)}{\sum_k \exp(\beta_i' x_k)} f(\beta_i|\theta)\, d\beta_i \end{aligned}$$

2.2 Assume for simplicity that $\xi_j = 0$. Estimate the parameters of the model (including σ_1 and a constant) using a consistent estimator.

First, we load the original dataset for further manipulation.

```
dataset <- read_excel(path = "dataset_ps1.xlsx") %>%
  mutate(market = as.factor(market),
         product = as.factor(product))
```

And for our starting guesses for the parameters, we will use the IV estimates from question 1.

The following code manually implements the BLP estimation procedure, using the gmm package. To do so, a gmm_fn(.) function is defined based on the 3 step procedure described in class.

```
# Add the outside option
outside_option <- dataset %>%
  group_by(market) %>%
  summarize(share = 1 - sum(share), .groups = 'drop') %>%
  mutate(product = "outside",
         x1 = 0, x2 = 0, x3 = 0, x4 = 0, price = 0,
         iv1 = 0, iv2 = 0, iv3 = 0, iv4 = 0, iv5 = 0, iv6 = 0)

# Combine the original dataset with the outside option
dataset_with_outside <- bind_rows(dataset, outside_option) %>%
  mutate(product = as.factor(product))

# Load initial parameter guesses from iv_model coefficient
intercept <- iv_model$coefficients["(Intercept)"]
alpha <- iv_model$coefficients["price"]
beta <- iv_model$coefficients[c("x1", "x2", "x3", "x4")]
sigma <- 0.1  # Initial value for sigma

# Set up initial parameters for GMM estimation
initial_params <- c(intercept = intercept, alpha = alpha, beta = beta, sigma = sigma)

set.seed(123)  # For reproducibility

# Set up constants
num_consumers <- 500
tolerance <- 1e-10  # Set a convergence tolerance
max_iterations <- 10000  # Maximum number of iterations

# Generate consumer taste shocks
taste_shocks <- dataset_with_outside %>%
  distinct(market) %>%
  mutate(consumer_id = list(1:num_consumers)) %>%
  unnest(consumer_id) %>%
  mutate(v_i = rnorm(n()))

# Expand dataset with product-consumer-market combinations
expanded_dataset <- crossing(
  dataset_with_outside %>% select(market, product),
  consumer_id = unique(taste_shocks$consumer_id)
) %>%
  left_join(taste_shocks, by = c("market", "consumer_id")) %>%
  left_join(dataset_with_outside, by = c("market", "product"))

# Define the GMM function
gmm_fn <- function(params, data, max_iterations = 10000, tolerance = 1e-6) {

  # Extract parameters
```

```r
intercept <- params["intercept.(Intercept)"]
alpha <- params["alpha.price"]
beta <- params[grep("^beta", names(params))]
sigma <- params["sigma"]

# Initialize delta based on the current parameter values
data <- data %>%
  mutate(delta = intercept + alpha * price + beta[1] * x1 + beta[2] * x2 + beta[3] * x3 + beta[4] * x4
         mu_i = sigma * v_i * x4)

# Main iteration loop to update delta
for (iteration in 1:max_iterations) {

  # Calculate expected utility
  data <- data %>%
    mutate(exp_utility = exp(delta + mu_i)) %>%
    group_by(market, consumer_id) %>%
    mutate(total_exp_utility = sum(exp_utility)) %>%
    ungroup() %>%
    mutate(div = exp_utility / total_exp_utility) %>%
    group_by(market, product) %>%
    mutate(s_jm = sum(div) / num_consumers) %>%
    ungroup()

  # Calculate the new delta values
  data <- data %>%
    mutate(delta_new = delta + log(share) - log(s_jm))  # Update delta

  # Check for NA values in delta_new and delta
  if (any(is.na(data$delta_new))) {
    cat("NA detected in delta_new values. Exiting iteration.\n")
    break
  }

  # Calculate the maximum difference for convergence check
  delta_diff <- max(abs(data$delta_new - data$delta), na.rm = TRUE)

  # Check for convergence
  if (delta_diff < tolerance) {
    # cat("Converged in", iteration, "iterations.\n")
    # cat("Converged with", params, "\n")
    break  # Exit the loop if converged
  }

  # Update delta for the next iteration
  data$delta <- data$delta_new
}

# Define the matrix of instruments (Z)
Z <- as.matrix(data[, c("iv1", "iv2", "iv3", "iv4", "iv5", "iv6")])
```

```r
  # Compute residuals based on the final delta
  X <- as.matrix(data[, c("price", "x1", "x2", "x3", "x4")])
  residuals <- data$delta - X %*% c(alpha, beta)

  # Calculate the moment conditions
  moments <- c(residuals) * Z

  # Return the average of the moments across observations
  return(colMeans(moments))
}

# Perform GMM estimation
gmm_results <- gmm(
  g = gmm_fn,
  x = expanded_dataset,
  t0 = initial_params,
  vcov = 'HAC',
  method = 'Nelder-Mead',
  control = list(reltol = 1e-25, maxit = 100000)
)
```

```
## Warning in FinRes.baseGmm.res(z, Model_info): The covariance matrix of the
## coefficients is singular
```

The model results are

```
## intercept.(Intercept)          alpha.price             beta.x1
##           -2.88347845          -71.16020851           1.47127215
##               beta.x2              beta.x3             beta.x4
##            1.68081209           2.15183723           4.72792619
##                 sigma
##            0.02315001
```

2.3 Compute own- and cross-price elasticities for each good in market one. Compare with the elasticities obtained in 1.5.

```r
# Constants
alpha <- coefs['alpha.price']
beta <- coefs[3:6]
sigma <- coefs['sigma']

# Filter data for market 1
market_data <- expanded_dataset %>%
  filter(market == 1, product != "outside")

# Calculate individual choice probabilities s_ij for each product and consumer in market 1
market_data <- market_data %>%
  group_by(consumer_id) %>%
  mutate(
    delta = intercept + alpha * price + beta[1] * x1 + beta[2] * x2 + beta[3] * x3 + beta[4] * x4,
    mu = sigma * v_i * x4,
    exp_delta_mu = exp(delta + mu),
```

```
    total_exp_utility = sum(exp_delta_mu),
    s_ij = exp_delta_mu / total_exp_utility
  ) %>%
  ungroup()

# Calculate average market share for each product (s_j)
average_shares <- market_data %>%
  group_by(product) %>%
  summarize(s_j = mean(s_ij))

# Sort products in ascending order
products <- sort(as.numeric(unique(market_data$product)))

# Initialize an empty matrix to store elasticities
elasticity_matrix <- matrix(0, nrow = length(products), ncol = length(products),
                            dimnames = list(products, products))

# Calculate own- and cross-price elasticities
for (j in products) {
  # Filter data for product j
  product_data_j <- market_data %>% filter(product == j)
  s_j <- average_shares %>% filter(product == j) %>% pull(s_j)
  p_j <- product_data_j$price[1]  # Assuming price is constant across consumers for each product

  # Calculate own-price elasticity for product j
  elasticity_matrix[j, j] <- mean((p_j / s_j) * alpha * product_data_j$s_ij * (1 - product_data_j$s_ij))

  # Calculate cross-price elasticities for product j with respect to other products k
  for (k in products[products != j]) {
    product_data_k <- market_data %>% filter(product == k)
    p_k <- product_data_k$price[1]  # Price of product k
    elasticity_matrix[j, k] <- -mean((p_k / s_j) * alpha * product_data_j$s_ij * product_data_k$s_ij)
  }
}
```

Comparing the new results to those observed in 1.5, the IAA is not valid anymore.

	1	2	3	4	5	6	7	8	9	10
1	-0.347177198	0.005299763	0.003086103	0.003082090	0.001784594	0.004257424	0.007630117	0.093878490	0.052453340	0.033284590
2	0.004048745	-0.288637041	0.003086080	0.003082065	0.001784572	0.004257394	0.007630063	0.093878900	0.052453390	0.033284560
3	0.004048802	0.005299800	-0.416630548	0.003082177	0.001784673	0.004257530	0.007630307	0.093877040	0.052453160	0.033284700
4	0.004048810	0.005299806	0.003086196	-0.381573431	0.001784686	0.004257548	0.007630339	0.093876790	0.052453120	0.033284720
5	0.004048858	0.005299845	0.003086282	0.003082286	-0.581821559	0.004257663	0.007630543	0.093875240	0.052452930	0.033284830
6	0.004048799	0.005299798	0.003086178	0.003082172	0.001784668	-0.434034241	0.007630296	0.093877120	0.052453170	0.033284690
7	0.004048799	0.005299797	0.003086178	0.003082172	0.001784668	0.004257524	-0.615454194	0.093877130	0.052453170	0.033284690
8	0.004048677	0.005299697	0.003085960	0.003081933	0.001784453	0.004257233	0.007629776	-0.658280020	0.052453670	0.033284390
9	0.004048713	0.005299727	0.003086024	0.003082004	0.001784516	0.004257319	0.007629930	0.093879910	-0.725158920	0.033284480
10	0.004048734	0.005299744	0.003086061	0.003082043	0.001784552	0.004257368	0.007630016	0.093879250	0.052453430	-0.842938800

Table 1: Elasticity Matrix for first 10 goods in market 1