Industrial Organization II: Problem Set 1

Arjun Gopinath and Clara Kyung

February 6, 2022

Question 1

1)

Define $\mu = \ln\left[\sum_{i=1}^n \exp\left(\frac{\mu_i}{\sigma}\right)\right]$. Then

$$\begin{aligned}
\Pr\{Y \leq x\} &= \Pr\{\bigcap_{i=1}^n (X_i \leq x)\} \\
&= \prod_{i=1}^n \Pr\{X_i \leq x; \mu_i, \sigma\} && \text{by indep.} \\
&= \exp\left\{-\sum_{i=1}^n \exp\left(-\frac{x-\mu_i}{\sigma}\right)\right\} \\
&= \exp\left\{-\exp\left(\frac{-x}{\sigma}\right)\sum_{i=1}^n \exp\left(\frac{\mu_i}{\sigma}\right)\right\} \\
&= \exp\left\{-\exp\left(-\frac{x-\mu\sigma}{\sigma}\right)\right\} && \text{by def'n of } \mu \\
&\sim T1EV(\mu\sigma, \sigma)
\end{aligned}$$

2)

(I skip some steps in this derivation because it is too much typing!)

$$
\begin{aligned}
\Pr\{X - Y \le z\} &= \Pr\{X \le Y + z\} \\
&= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{Y+z} f_X(x)dx \right) f_Y(y)dy \\
&= \int_{-\infty}^{\infty} f_Y(y) F_X(y+z)dy \\
&= \int_{-\infty}^{\infty} \frac{1}{\sigma} \exp\left\{ -\frac{y-\mu_Y}{\sigma} \right\} \exp\left\{ -\exp\left(-\frac{y-\mu_Y}{\sigma} \right) \left(1 + \exp\left(-\frac{z+\mu_Y-\mu_X}{\sigma} \right) \right) \right\} dy \\
&\qquad \text{define } a \equiv 1 + \exp\left(-\frac{z+\mu_Y-\mu_X}{\sigma} \right) \\
&= \int_{-\infty}^{\infty} \frac{1}{\sigma} \exp\left\{ -\frac{y-\mu_Y}{\sigma} \right\} \exp\left\{ -\exp\left(-\frac{y-\mu_Y}{\sigma} \right) a \right\} dy \\
&= \frac{1}{a} \int_{-\infty}^{\infty} a \frac{1}{\sigma} \exp\left\{ -\frac{y-\mu_Y}{\sigma} \right\} \exp\left\{ -a \exp\left(-\frac{y-\mu_Y}{\sigma} \right) \right\} dy
\end{aligned}
$$

We now make a change of variables[1]: Define $u \equiv a \exp\left(-\frac{y-\mu_Y}{\sigma}\right)$. Then $du = \frac{-a}{\sigma} \exp\left(-\frac{y-\mu_Y}{\sigma}\right)$, and as $y \to -\infty$, $u \to \infty$, and $y \to \infty$, $u \to 0$. So, we may write the above as

$$
\begin{aligned}
\Pr\{X - Y \le z\} &= \frac{1}{a} \int_{\infty}^{0} -\exp\left\{ -a \exp\left(-\frac{y-\mu_Y}{\sigma} \right) \right\} \frac{a}{\sigma} \exp\left\{ -\frac{y-\mu_Y}{\sigma} \right\} dy \\
&= \frac{1}{a} \int_{\infty}^{0} -\exp(-u)du \\
&= \frac{1}{a} \int_{0}^{\infty} \exp(-u)du \\
&= \frac{1}{a} \\
&= \frac{\exp\left(-\frac{z-(\mu_X-\mu_Y)}{\sigma}\right)}{1 + \exp\left(-\frac{z-(\mu_X-\mu_Y)}{\sigma}\right)}
\end{aligned}
$$

which we recognize as the cdf of a Logistic$(\mu_X - \mu_Y, \sigma)$ random variable.

3)

For this derivation, first define $u_{-j} = \max_{k \ne j}\{\mu_k + \epsilon_k\}$ and $\mu_{-j} = \ln\left(\sum_{k \ne j} \exp(\mu_k)\right)$. By lemma 2.2.2 in the textbook, $u_{-j} \sim T1EV(\mu_{-j})$. So:

$$
\begin{aligned}
\Pr\{u_j > u_k, \forall k \ne j\} &= \Pr\{\mu_j + \epsilon_j > \max_{k \ne j}\{\mu_k + \epsilon_k\}\} \\
&= \Pr\{u_j \ge u_{-j}\} \\
&= \Pr\{u_{-j} - u_j \le 0\} \\
&= \frac{\exp(\mu_j)}{\exp(\mu_{-j}) + \exp(\mu_j)} && \text{by lemma 2.2.3 in textbook} \\
&= \frac{\exp(\mu_j)}{\sum_k \exp(\mu_k)}
\end{aligned}
$$

[1]Thanks to Conroy and Feng for help with this part.

4)

i)

We have that the latent utilities are independently distributed according to $u_{ij} \sim T1EV(\alpha(y_i - p_j), 1)$. Therefore, from theorem 2.2.1 in the textbook, we have

$$s_{ij} = \frac{\exp(\alpha(y_i - p_j))}{\sum_{k\in\mathcal{J}} \exp(\alpha(y_i - p_k))} = \frac{\exp(-\alpha p_j)}{\sum_{k\in\mathcal{J}} \exp(-\alpha p_k)}$$

and

$$\frac{\partial s_j(i)}{\partial y_i} = 0$$

i.e., the demand elasticity with respect to income is 0. This is because income enters utility linearly, so y_i just cancels out in the probability.

ii)

Maket share of product j:

$$\begin{aligned}
s_j &= \int_{\mathcal{Y}} s_{ij} dF(y_i) \\
&= \int_{\mathcal{Y}} \frac{\exp(-\alpha p_j)}{\sum_{k\in\mathcal{J}} \exp(-\alpha p_k)} dF(y_i) \\
&= \frac{\exp(-\alpha p_j)}{\sum_{k\in\mathcal{J}} \exp(-\alpha p_k)} \\
&= s_{ij}
\end{aligned}$$

The own-price elasticity is

$$\frac{\partial s_j}{\partial p_j}\frac{p_j}{s_j} = -\alpha s_j(1-s_j)\frac{p_j}{s_j} = -\alpha(1-s_j)p_j$$

and the cross-price elasticity is

$$\frac{\partial s_j}{\partial p_k}\frac{p_k}{s_j} = \alpha s_j s_k \frac{p_k}{s_j} = \alpha s_k p_k$$

The own and cross-price elasticities depend only on prices and market shares. This is probably not reasonable. Own-price elasticity should realistically be affected by the utility that can be gained from other products (are there close substitutes or complements?). And as discussed in class through the BMW-Mercedes-Kia example, this expression for cross-price elasticity places strong (and unrealistic) restrictions on substitution patterns.

iii)

Assume $\beta_i = \beta$

Remark that in this case, heterogeneity between consumers is due to y_i only. However, y_i does not affect how consumers make their choices, so all consumers will choose the same good. One good will have a market share of 1 and the rest 0.

The probability that consumer i chooses product j:

$$\begin{aligned} s_{ij} &= \Pr\{\alpha(y_i - p_j) + \beta x_j > \max_{k \neq j} \alpha(y_i - p_k) + \beta x_k\} \\ &= \prod_{k \neq j} \Pr\{\beta(x_j - x_k) > \alpha(p_j - p_k)\} \\ &= \prod_{k \neq j} \mathbb{1}\left[\beta(x_j - x_k) > \alpha(p_j - p_k)\right] \end{aligned}$$

The market share of product j:

$$s_j = s_{ij} = \prod_{k \neq j} \mathbb{1}\left[\beta(x_j - x_k) > \alpha(p_j - p_k)\right]$$

because, as explained above, all consumers make the same choice.

Assume $\beta_i \sim Unif[0, \overline{\beta}]$, iid
The probability that consumer i chooses product j:

$$\begin{aligned} s_{ij} &= \Pr\{\alpha(y_i - p_j) + \beta_i x_j > \max_{k \neq j} \alpha(y_i - p_k) + \beta_i x_k\} \\ &= \prod_{k \neq j} \Pr\{\beta_i(x_j - x_k) > \alpha(p_j - p_k)\} \\ &= \prod_{k \neq j} \left(1 - \Pr\left\{\beta_i \leq \alpha \frac{p_j - p_k}{x_j - x_k}\right\}\right) \\ &= \prod_{k \neq j} \left(1 - \frac{\alpha}{\overline{\beta}} \frac{p_j - p_k}{x_j - x_k}\right) \end{aligned}$$

Notice that once again, s_{ij} does not depend on i. This is because the unobserved heterogeneity (β_i) is iid. So the market share of product j is

$$s_j = s_{ij} = \prod_{k \neq j} \left(1 - \frac{\alpha}{\overline{\beta}} \frac{p_j - p_k}{x_j - x_k}\right)$$

The own-price elasticity is:

$$\begin{aligned} \frac{\partial s_j}{\partial p_j} &= \frac{\partial}{\partial p_j} \prod_{k \neq j} \left(1 - \frac{\alpha}{\overline{\beta}} \frac{p_j - p_k}{x_j - x_k}\right) \\ &= \prod_{l \neq j} \left[-\frac{\alpha}{\overline{\beta}} \frac{1}{x_j - x_l} \prod_{k \neq j, l} \left(1 - \frac{\alpha}{\overline{\beta}} \frac{p_j - p_k}{x_j - x_k}\right)\right] \\ &= \prod_{l \neq j} \left[\frac{-\frac{\alpha}{\overline{\beta}} \frac{1}{x_j - x_l}}{1 - \frac{\alpha}{\overline{\beta}} \frac{p_j - p_l}{x_j - x_l}} s_j\right] \end{aligned}$$

and the cross-price elasticity is

$$\begin{aligned}
\frac{\partial s_j}{\partial p_l} &= \frac{\partial}{\partial p_l} \prod_{k \neq j} \left(1 - \frac{\alpha}{\beta} \frac{p_j - p_k}{x_j - x_k}\right) \\
&= \frac{\frac{\alpha}{\beta} \frac{1}{x_j - x_l}}{1 - \frac{\alpha}{\beta} \frac{p_j - p_l}{x_j - x_l}} s_j
\end{aligned}$$

Here the cross-price elasticities depend on how close the products are in characteristics space $(x_j - x_l)$. This is more reasonable than the result in (ii). Consider the BMW, Mercedes-Benz, and Kia example again. Suppose $i =$BMW, $j =$Mercedes-Benz, and $k =$Kia. Write the cross-price elasticities:

$$\begin{aligned}
\frac{\partial s_i}{\partial p_j} \frac{p_j}{s_i} &= \frac{\frac{\alpha}{\beta} \frac{p_j}{x_i - x_j}}{1 - \frac{\alpha}{\beta} \frac{p_i - p_j}{x_i - x_j}} \\
\frac{\partial s_j}{\partial p_k} \frac{p_k}{s_j} &= \frac{\frac{\alpha}{\beta} \frac{p_k}{x_j - x_k}}{1 - \frac{\alpha}{\beta} \frac{p_j - p_k}{x_j - x_k}}
\end{aligned}$$

Because of our assumption that $\frac{p_i - p_j}{x_i - x_j} \geq \frac{p_j - p_k}{x_j - x_k}$ whenever $x_i \geq x_j \geq x_k$, and because $\frac{p_j}{x_i - x_j} \geq \frac{p_k}{x_j - x_k}$ very likely holds (since BMW and Mercedes are closer in characteristics space than Mercedes-Benz and Kia, and the price of Mercedes-Benz is likely higher than Kia), we have that

$$\frac{\partial s_i}{\partial p_j} \frac{p_j}{s_i} \geq \frac{\partial s_j}{\partial p_k} \frac{p_k}{s_j}$$

i.e., demand for BMWs is more sensitive to changes in the price of Mercedes-Benz than demand for Mercedes-Benz is to changes in the price of Kia. We do not get this result in (ii), where cross-price elasticities depended only on market shares and prices.

iv)

Now we have $u_{ij} = \alpha(y_i - p_j) + \beta_i x_j + \nu_{ij}$, where $\beta_i \sim^{iid} F(.)$ and $\nu_{ij} \sim^{iid} T1EV(0, 1)$.

The probability that i chooses j is

$$s_{ij} = \frac{\exp(-\alpha p_j + \beta_i x_j)}{\sum_k \exp(-\alpha p_k + \beta_i x_k)}$$

and the market share of product j is

$$\begin{aligned}
s_j &= \int s_{ij} dF(\beta_i) \\
&= \int \frac{\exp(-\alpha p_j + \beta_i x_j)}{\sum_k \exp(-\alpha p_k + \beta_i x_k)} dF(\beta_i)
\end{aligned}$$

The own price elasticity is

$$\begin{aligned}
\frac{\partial s_j}{\partial p_j} \frac{p_j}{s_j} &= \frac{p_j}{s_j} \int \frac{\partial}{\partial p_j} \frac{\exp(-\alpha p_j + \beta_i x_j)}{\sum_k \exp(-\alpha p_k + \beta_i x_k)} dF(\beta_i) \\
&= \frac{p_j}{s_j} \int -\alpha s_{ij}(1 - s_{ij}) dF(\beta_i)
\end{aligned}$$

and the cross-price elasticity is

$$
\begin{aligned}
\frac{\partial s_j}{\partial p_k}\frac{p_k}{s_j} &= \frac{p_k}{s_j}\int \frac{\partial}{\partial p_k}\frac{\exp(-\alpha p_j + \beta_i x_j)}{\sum_k \exp(-\alpha p_k + \beta_i x_k)} dF(\beta_i) \\
&= \frac{p_k}{s_j}\int \alpha s_{ij} s_{ik} dF(\beta_i)
\end{aligned}
$$

The price elasticities are now determined in part by the distribution of β_i in the population. Since the price elasticities do not depend solely on market shares and prices, we no longer have independence of irrelevant alternatives (IIA) at the market level, unlike in (ii).

v)

Rewrite W as a function of the market share of the outside good s_0:

We have that $u_{ij} \sim T1EV(\alpha(y_i - p_j), 1)$, independently distributed. By lemma 2.2.2 in the textbook,

$$
\mathbb{E}\left[\max_j u_{ij}\right] = \ln\left(\sum_j \exp(\alpha(y_i - p_j))\right) + \gamma
$$

We also have that

$$
s_0 = \frac{1}{\sum_j \exp(\alpha(y_i - p_j))} \Rightarrow \sum_j \exp(\alpha(y_i - p_j)) = \frac{1}{s_0}
$$

So,

$$
W \equiv E\max_j u_{ij} = \ln\left(\frac{1}{s_0}\right) + \gamma
$$

What happens to W when a new product $J+1$ is introduced in the market?:

Define $W^+ = \mathbb{E}\left[\max\{\max_{j\in\mathcal{J}} u_{ij}, u_{i,J+1}\}\right]$. By lemma 2.2.2 in the textbook, and using our result that $\sum_j^J \exp(\alpha(y_i - p_j)) = \frac{1}{s_0}$ we have that

$$
\begin{aligned}
\mathbb{E}\left[\max\{\max_{j\in\mathcal{J}} u_{ij}, u_{i,J+1}\}\right] &= \ln\left(\sum_j^{J+1} \exp(\alpha(y_i - p_j))\right) + \gamma \\
&= \ln\left(\sum_j^{J} \exp(\alpha(y_i - p_j)) + \exp(\alpha(y_i - p_{J+1}))\right) + \gamma \\
&= \ln\left(\frac{1}{s_0} + \exp(\alpha(y_i - p_{J+1}))\right) + \gamma
\end{aligned}
$$

Then $W^+ - W = \ln(1 + s_{i0}\exp(\alpha(y_i - p_{J+1})) \geq 0$. W increases when a new good is introduced to the market. This is what we'd expect, because no consumer is made worse off by having an additional option to choose from.

Question 2

(i)

β reflects the effect of product characteristics on utility that is common to all individuals. Γ reflects the effect of product characteristics on utility that varies by demographics.

(ii)

First, define

$$s_{ij} := \Pr\{i \text{ chooses } j\} = \frac{\exp(\delta_j + d_i'\Gamma X_j)}{\sum_{k\in\mathcal{J}} \exp(\delta_k + d_i'\Gamma X_k)}$$

The log-likelihood of the data is then

$$\log L(y;\delta,\Gamma) = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \mathbf{1}\{i \text{ chooses } j\} \left(\delta_j + d_i'\Gamma X_j - \log(\sum_{k\in\mathcal{J}} \exp(\delta_k + d_i'\Gamma X_k))\right)$$

(iii)

The FOC of the log-likelihood with respect to δ_j is

$$\begin{aligned} 0 &= \sum_{i\in\mathcal{I}} \mathbf{1}\{i \text{ chooses } j\} - \sum_{i\in\mathcal{I}} s_{ij} \sum_{k\in\mathcal{J}} \mathbf{1}\{i \text{ chooses } k\} \\ &= \sum_{i\in\mathcal{I}} \mathbf{1}\{i \text{ chooses } j\} - \sum_{i\in\mathcal{I}} s_{ij} \\ &= \sum_{i\in\mathcal{I}} (\mathbf{1}\{i \text{ chooses } j\} - s_{ij}) \end{aligned}$$

Interpretation: We can rewrite the above FOC as $\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} s_{ij} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \mathbf{1}\{i \text{ chooses } j\}$. The LHS is the average probability that an individual chooses good j. The RHS is the proportion of individuals who choose product j. The LHS is a function of the δ_j's and the RHS is what we observe. We want to pick δ_j's such that these two quantities are equal.

The FOC of the log-likelihood with respect to Γ is

$$0 = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \mathbf{1}\{i \text{ chooses } j\} d_i X_j' (1 - s_{ij})$$

Interpretation: We can rewrite the FOC as
$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \mathbf{1}\{i \text{ chooses } j\} d_i X_j' = \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \mathbf{1}\{i \text{ chooses } j\} d_i X_j' s_{ij}$. On the LHS, we are summing all the $d_i X_j'$ for which individual i chose product j. On the RHS, we are summing all the $d_i X_j'$ for which individual i chose product j, scaled by the probability that individual i chooses j. I must admit I am not 100% sure how to interpret the FOC for Γ...

(iv)

See attached Jupyter Notebook for MLE code (done in Python). Table 1 contains the MLE results.

	delta		Gamma
δ_1	1.03	Γ_{11}	0.71
δ_2	0.17	Γ_{12}	0.10
δ_3	0.42	Γ_{13}	0.62
δ_4	0.41	Γ_{21}	0.51
δ_5	2.18	Γ_{22}	0.22
δ_6	1.62	Γ_{23}	0.77
δ_7	-0.23		.
δ_8	1.44		.
δ_9	1.10		.
δ_{10}	1.19		.
δ_{11}	1.31		.
δ_{12}	1.80		.
δ_{13}	0.96		.
δ_{14}	-0.93		.
δ_{15}	0.58		.
δ_{16}	1.40		.
δ_{17}	0.73		.
δ_{18}	1.09		.
δ_{19}	0.64		.
δ_{20}	0.46		.
δ_{21}	1.89		.
δ_{22}	1.41		.
δ_{23}	1.35		.
δ_{24}	2.95		.
δ_{25}	0.89		.
δ_{26}	1.45		.
δ_{27}	-1.89		.
δ_{28}	0.42		.
δ_{29}	3.21		.
δ_{30}	1.67		.

Table 1: Maximum Likelihood Estimates of δ and Γ coefficients

(v)

We defined $\delta_j = x_j'\beta + \xi_j$, where x_j are observed and ξ_j are unobserved product characteristics. One moment condition (albeit an unrealistic one) that we could assume is $\mathbb{E}[x_j\xi_j] = 0$. If this moment condition holds, we could regress δ_j on x_j using OLS to consistently estimate β.

(vi)

The table below shows the β coefficients estimated from OLS.

	(1)
x.1	-0.07
	(0.11)
x.2	0.82***
	(0.11)
x.3	0.18**
	(0.07)
N	30
R^2	0.88

Table 2: OLS estimates of β coefficients

In [1]:
```
import numpy as np
import pandas as pd
import pyblp as blp
import torch
from torch.autograd import Variable
import torch.optim as optim
from linearmodels.iv import IV2SLS
from HomogenousDemandEstimation import HomDemEst

blp.options.digits = 2
blp.options.verbose = False
nax = np.newaxis
```

Exercise 3

The file `ps1_ex3.csv` contains aggregate data on a large number $T = 1000$ of markets in which $J = 6$ products compete between each other together with an outside good $j = 0$. The utility of consumer i is given by:

$$\begin{aligned} u_{ijt} &= -\alpha p_{jt} + \mathbf{x}_{jt}\boldsymbol{\beta} + \xi_{jt} + \epsilon_{ijt} \quad j = 1, \ldots, 6 \\ u_{i0t} &= \epsilon_{i0t} \end{aligned}$$

where p_{jt} is the price of product j in market t, $\mathbf{x}_{jt}$ is an observed product characteristic (including a constant), ξ_{jt} is an unobserved product characteristic and ϵ_{ijt} is i.i.d T1EV $(0, 1)$. Our goal is to to estimate demand parameters $(\alpha, \boldsymbol{\beta})$ and perform some counterfactual exercises.

In [2]:
```
# Load the dataset.
data_ex3 = pd.read_csv('ps1_ex3.csv')
num_prod = data_ex3.Product.max()
num_T = data_ex3.market.max()
```

Part 1

Assuming that the variable z in the dataset is a valid instrument for prices, write down the moment condition that allows you to consistently estimate (α, β) and obtain an estimate for both parameters.

Under the T1EV assumption, we can derive the CCPs which corresponds to the predicted market share for product j at time t. This can be approximated from the data using the observed market share s_{jt}.

$$\Pr(i \text{ chooses } j \text{ at time } t) = \frac{\exp\left(-\alpha p_{jt} + \mathbf{x}'_{jt}\boldsymbol{\beta} + \xi_{jt}\right)}{\sum_{k \in \mathcal{J}_t} \exp\left(-\alpha p_{kt} + \mathbf{x}'_{kt}\boldsymbol{\beta} + \xi_{kt}\right)} \approx s_{jt}$$

We can invoke the normalization assumption on u_{i0} and take the logarithm of the share ratio s_{jt}/s_{0t} to obtain

$$\ln\left(\frac{s_{jt}}{s_{0t}}\right) = -\alpha p_{jt} + \mathbf{x}'_{jt}\boldsymbol{\beta} + \xi_{jt}$$

In [3]:

```
# Create outside option shares and merge into dataset.
share_total = data_ex3.groupby(['market'])['Shares'].sum().reset_index()
share_total.rename(columns={'Shares': 's0'}, inplace=True)
share_total['s0'] = 1 - share_total['s0']
data_ex3 = pd.merge(data_ex3, share_total, on='market')

# Create natural log of share ratios
data_ex3['s_ratio'] = np.log(data_ex3['Shares']/data_ex3['s0'])
```

Given that z_{jt} is a relevant instrument for p_{jt} and that $\mathbf{x}_{jt}$ is exogenous, we can impose the conditional exogeneity restriction

$$\mathbb{E}\left[\xi_{jt} \mid \mathbf{x}_{jt}, z_{jt}\right] = 0$$

in order to estimate α and $\boldsymbol{\beta}$. Using the Law of Iterated Expectations, we can conclude that

$$\mathbb{E}\left[\begin{pmatrix}\mathbf{x}_{jt} \\ z_{jt}\end{pmatrix}\xi_{jt}\right] = \mathbb{E}\left[\begin{pmatrix}\mathbf{x}_{jt} \\ z_{jt}\end{pmatrix}\left\{\ln\left(\frac{s_{jt}}{s_{0t}}\right) + \alpha p_{jt} - \mathbf{x}'_{jt}\boldsymbol{\beta}\right\}\right] = \begin{pmatrix}0 \\ 0\end{pmatrix}$$

Given that 3 moment conditions across all products and markets, we are exactly identifying α and β.

GMM provides the minimizer corresponding to a quadratic loss function with 3 moments.

$$\begin{pmatrix}\widehat{\alpha} \\ \widehat{\boldsymbol{\beta}}\end{pmatrix} \in \underset{\begin{pmatrix}\alpha \\ \beta\end{pmatrix}}{\arg\min}\left[\frac{1}{T \times J}\sum_{t}\sum_{j} x_{jt}\left\{\ln\left(\frac{s_{jt}}{s_{0t}}\right) + \alpha p_{jt} - \mathbf{x}'_{jt}\boldsymbol{\beta}\right\}\right]$$

I perform the two-step procedure to obtain the efficient GMM estimator of the model parameters.

In [4]:

```
est = HomDemEst(data_dict={
    'Data': data_ex3,
    'Choice Column': 'Product',
    'Market Column': 'market',
    'Log Share Ratio Column': 's_ratio',
    'Endogenous Columns': ['Prices'],
    'Exogenous Columns': ['x'],
    'Instrument Columns': ['z'],
    'Add Constant': True
})

results = est.run_gmm()
```

In [5]:

```
results['Coefficients']
```

Out[5]:
```
tensor([ 0.7289,  0.3047, -0.4675], dtype=torch.float64)
```

In [6]:
```
torch.sqrt(torch.diag(results['Covariance Matrix']))
```

Out[6]:
```
tensor([0.1989, 0.0084, 0.0633], dtype=torch.float64)
```

We find the following estimates for α and β.

Coefficient	Estimate	Std. Error
Constant	0.7289	0.1989
Prices	0.4675	0.0633
Product Characteristic	0.3047	0.0084

In [7]:
```
data_ex3['const'] = 1

iv = IV2SLS(dependent=data_ex3['s_ratio'],
       exog=data_ex3[['const', 'x']],
       endog=data_ex3['Prices'],
       instruments=data_ex3['z']).fit(cov_type='unadjusted')

print(iv.summary)
```

```
                          IV-2SLS Estimation Summary
==============================================================================
Dep. Variable:                s_ratio   R-squared:                      0.0018
Estimator:                    IV-2SLS   Adj. R-squared:                 0.0015
No. Observations:                6000   F-statistic:                    1575.0
Date:                Sun, Feb 06 2022   P-value (F-stat)                0.0000
Time:                        11:46:39   Distribution:                  chi2(2)
Cov. Estimator:            unadjusted

                             Parameter Estimates
==============================================================================
            Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
------------------------------------------------------------------------------
const          0.7289     0.1944     3.7487     0.0002      0.3478      1.1100
x              0.3047     0.0083     36.553     0.0000      0.2883      0.3210
Prices        -0.4675     0.0618    -7.5651     0.0000     -0.5886     -0.3464
==============================================================================

Endogenous: Prices
Instruments: z
Unadjusted Covariance (Homoskedastic)
Debiased: False
```

Part 2

We know that the elasticities for homogenous demand are given by

$$\varepsilon_{jk,t} = \begin{cases} -\alpha p_{j,t}\left(1-\pi_{j,t}\right) & \text{if } j=k \\ \alpha p_{k,t}\pi_{k,t} & \text{otherwise} \end{cases}$$

In [8]:

```
α = -np.array(results['Coefficients'])[-1]
```

In [9]:

```
# Compute the own and cross-price elasticity for each market and pair of product
data_ex3['own'] = -α * data_ex3['Prices'] * (1 - data_ex3['Shares'])
data_ex3['cross'] = α * data_ex3['Prices'] * data_ex3['Shares']
e_mean = data_ex3.groupby(['Product'])[['own', 'cross']].mean()

# Generate matrix of average elasticities.
e_mat = np.tile(e_mean['cross'], (num_prod, 1))
np.fill_diagonal(e_mat, e_mean['own'])

# Convert it to a dataframe.
prod_list = list(map(str, range(1, num_prod+ 1)))
e_mat = pd.DataFrame(e_mat, index=prod_list, columns=prod_list)
```

j/k	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6
Product 1	-1.249624	0.323002	0.128738	0.127154	0.125510	0.129465
Product 2	0.321076	-1.251242	0.128738	0.127154	0.125510	0.129465
Product 3	0.321076	0.323002	-1.289146	0.127154	0.125510	0.129465
Product 4	0.321076	0.323002	0.128738	-1.293869	0.125510	0.129465
Product 5	0.321076	0.323002	0.128738	0.127154	-1.291425	0.129465
Product 6	0.321076	0.323002	0.128738	0.127154	0.125510	-1.290799

The results above show that the own-price elasticities are pretty consistent across the various products, and that the magnitude of the cross-price elasticities is lower than the corresponding own-price elasticities.

Part 3

To back out the marginal costs for producing each product in market t, we must first construct the conduct matrix $\mathbf{H}_t$ corresponding to the entire choice set. Since we assume that firms are single-product producers in all markets, the conduct matrix will be an identity matrix of dimension 6. Furthermore, we will need the matrix $\mathbf{\Omega}_t$ containing partial derivatives $-\partial q_{kt}/\partial p_{jt}$ multiplied by the corresponding entries $\mathbf{H}_t$. However, since $\mathbf{H}_t$ is an identity matrix, we only need to compute the diagonal entries of $\mathbf{\Omega}_t$ as all the off-diagonals entries will equal 0. Therefore, we can back out the following expression for diagonal element j of $\mathbf{\Omega}_t$ assuming that N_t represents the total size of the market.

$$\Omega_{jj,t} = -\frac{\partial q_{jt}}{\partial p_{jt}} = -\varepsilon_{jj,t}\frac{q_{jt}}{p_{jt}} = -\varepsilon_{jj,t}\frac{s_{jt}}{p_{jt}}N_t$$

The firm's profit maximization problem yields the following FOC:

$$\mathbf{p}_t - \mathbf{mc}_t \;=\; \mathbf{\Omega}_t^{-1}\mathbf{q}(\mathbf{p}_t) \;=\; \mathbf{\Omega}_t^{-1}\mathbf{s}(\mathbf{p}_t)N_t$$

Since $\mathbf{\Omega}$ is a diagonal matrix, we can back out the marginal cost of each product j in market t independently of the other products.

$$\begin{aligned} p_{jt} - \text{mc}_{jt} \;&=\; \Omega_{jj,t}^{-1} s_{jt} N_t \\ \text{mc}_{jt} \;&=\; p_{jt} + \frac{1}{\varepsilon_{jj,t} N_t}\frac{p_{jt}}{s_{jt}} s_{jt} N_t \;=\; p_{jt}\left(1 + \frac{1}{\varepsilon_{jj,t}}\right) \end{aligned}$$

In [11]:

```
data_ex3['mc'] = data_ex3['Prices'] * (1 + 1/data_ex3['own'])
mc_avg = data_ex3.groupby(['Product'])['mc'].mean()
```

In [12]:

```
all_avg = data_ex3.groupby(['Product'])[['Prices', 'Shares', 'mc']].mean()
```

We obtain the following average (across markets) marginal cost for each product. They are highly positively correlated with the average (across markets) prices and shares.

Product	Average Price	Average Share	Average MC
1	3.35995	0.202451	0.667126
2	3.36753	0.203076	0.671897
3	3.03306	0.0903493	0.678684
4	3.03977	0.0889407	0.688906
5	3.03103	0.0881716	0.682632
6	3.03815	0.0906875	0.682689

Part 4

Suppose that product $j = 1$ exits the market, and marginal costs and product characteristics for the other products remain unchanged. Since the elasticities of demand for each product remain unchanged and given that all firms are single-product producers, the matrix Ω does not change for the remaining products, which implies that prices do not change for the products and average prices stay the same. To compute the new shares, we will recompute s_0 from the log share ratios for products $j = 2, \cdots, 6$. Under the new regime, we have that $\sum_{j=0,2}^{6} s_j = 1$. Therefore, we can write that

$$\sum_{j=2}^{6} \exp\log\frac{s_j}{s_0} = \frac{1-s_0}{s_0} \quad\Rightarrow\quad s_0 \;=\; \left(1 + \sum_{j=2}^{6}\frac{s_j}{s_0}\right)^{-1}$$

In [ ]:
```
data_new = data_ex3.query('Product != 1')
data_new.drop(columns=['s0'], inplace=True)
data_new['exp_s_ratio'] = np.exp(data_new['s_ratio'])

# Create sum of share ratios
new_s0 = data_new.groupby(['market'])['exp_s_ratio'].sum().reset_index()
new_s0.rename(columns={'exp_s_ratio': 's0'}, inplace=True)
new_s0.loc[:, 's0'] = 1 / (1 + new_s0.loc[:, 's0'])
data_new = pd.merge(data_new, new_s0, on='market')
data_new['Shares_new'] = data_new['exp_s_ratio'] * data_new['s0']
# data_new.drop(columns=['exp_s_ratio'])

data_new['Profit_old'] = (data_new['Prices'] - data_new['mc']) * data_new['Share
data_new['Profit_new'] = (data_new['Prices'] - data_new['mc']) * data_new['Share
data_new['Profit_change'] = data_new['Profit_new'] - data_new['Profit_old']
```

In [14]:
```
new_avg = data_new.groupby(['Product'])[['Prices', 'Shares_new', 'mc', 'Profit_o
```

Product	Average Price	Average Share	Average Change in Profits
2	3.36753	0.254303	0.139413
3	3.03306	0.113245	0.054411
4	3.03977	0.111318	0.053106
5	3.03103	0.110616	0.053617
6	3.03815	0.113617	0.054554

We see that all products (and correspondingly the firms that produce them) face an increase in profits, especially product 2. Consumers will face a reduction in welfare due to the increase in concentration of market power among the remaining firms.

In [14]:

In [1]:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import pyblp as blp
import torch
from torch.autograd import Variable
import torch.optim as optim
from linearmodels.iv import IV2SLS
from HomogenousDemandEstimation import HomDemEst
from GaussHermiteQuadrature import GaussHermiteQuadrature

blp.options.digits = 2
blp.options.verbose = False
nax = np.newaxis
```

The file `ps1_ex4.csv` contains aggregate data on $T = 600$ markets in which $J = 6$ products compete between each other together with an outside good $j = 0$. The utility of consumer i is given by:

$$\begin{aligned} u_{ijt} &= \widetilde{\mathbf{x}}_{jt}'\boldsymbol{\beta} + \xi_{jt} + \widetilde{\mathbf{x}}_{jt}'\boldsymbol{\Gamma}\boldsymbol{v}_i + \epsilon_{ijt} \quad j = 1, \ldots, 6 \\ u_{i0t} &= \epsilon_{i0t} \end{aligned}$$

where x_{jt} is a vector of observed product characteristics including the price, ξ_{jt} is an unobserved product characteristic, v_i is a vector of unobserved taste shocks for the product characteristics and ϵ_{ijt} is i.i.d T1EV $(0, 1)$. Our goal is to to estimate demand parameters $(\boldsymbol{\beta}, \boldsymbol{\Gamma})$ using the BLP algorithm. As you can see from the data there are only two characteristics $\widetilde{\mathbf{x}}_{jt} = \begin{pmatrix} p_{jt} & x_{jt} \end{pmatrix}$, namely prices and an observed measure of product quality. Moreover, there are several valid instruments $\mathbf{z}_{jt}$ that you will use to construct moments to estimate $(\boldsymbol{\beta}, \boldsymbol{\Gamma})$. Finally, you can assume that Γ is lower triangular e.g.,

$$\boldsymbol{\Gamma} = \begin{pmatrix} \gamma_{11} & 0 \\ \gamma_{21} & \gamma_{22} \end{pmatrix}$$

such that $\boldsymbol{\Gamma}\boldsymbol{\Gamma}' = \boldsymbol{\Omega}$ is a positive definite matrix and that v_i is a 2 dimensional vector of i.i.d random taste shocks distributed $\mathcal{N}(\mathbf{0}, \mathbf{I}_2)$.

In [2]:

```
# Load the dataset.
data_ex4 = pd.read_csv('ps1_ex4.csv')
data_ex4['const'] = 1.0     # Add a constant term

num_prod = data_ex4.choice.nunique()   # Number of products to choose from.
num_T = data_ex4.market.nunique()

# Create outside option shares and merge into dataset.
share_total = data_ex4.groupby(['market'])['shares'].sum().reset_index()
share_total.rename(columns={'shares': 's0'}, inplace=True)
share_total['s0'] = 1 - share_total['s0']
data_ex4 = pd.merge(data_ex4, share_total, on='market')

# Create natural log of share ratios
data_ex4['sr'] = np.log(data_ex4['shares']/data_ex4['s0'])
```

```python
# Create constant term
data_ex4['const'] = 1
```

The market shares can be expressed as a function of individual characteristics as shown below.

$$
\begin{aligned}
s_j &\simeq \mathbb{E}[\Pr(i \text{ Chooses } j)] \\
&= \int_{\mathbf{v}_i} \Pr(i \text{ Chooses } j) \,\mathrm{d}\, F\left(\mathbf{v}_i\right) \\
&= \int_{\mathbf{v}_i} \frac{\exp\left(\widetilde{\mathbf{x}}_{jt}'\boldsymbol{\beta} + \xi_{jt} + \widetilde{\mathbf{x}}_{jt}'\boldsymbol{\Gamma}\boldsymbol{v}_i\right)}{1 + \sum_{k\in\mathcal{J}_t}\exp\left(\widetilde{\mathbf{x}}_{kt}'\boldsymbol{\beta} + \xi_{kt} + \widetilde{\mathbf{x}}_{kt}'\boldsymbol{\Gamma}\boldsymbol{v}_i\right)} \mathrm{d}\, F\left(\mathbf{v}_i\right)
\end{aligned}
$$

However, due to the heterogeneity in individual preferences, we do not have a neat solution to back out the preference parameters from using logarithms of share-ratios.

In [3]:

```python
# Obtain initial guess for β using the homogenous model.

est = HomDemEst(data_dict={
    'Data': data_ex4,
    'Choice Column': 'choice',
    'Market Column': 'market',
    'Log Share Ratio Column': 'sr',
    'Endogenous Columns': ['p'],
    'Exogenous Columns': ['x'],
    'Instrument Columns': ['z1', 'z2', 'z3', 'z4', 'z5', 'z6'],
    'Add Constant': True
})

beta_guess = torch.tensor(np.array(est.one_step_gmm().detach()), dtype=torch.dou
beta_guess
```

Out[3]:

```
tensor([-1.9158,  0.7115, -0.3054], dtype=torch.float64)
```

In [4]:

```python
# Set parameters for the optimization procedure.
gamma = Variable(3 * torch.rand((2,2), dtype=torch.double), requires_grad=True)
beta = Variable(beta_guess, requires_grad=False)

print(gamma)
```

```
tensor([[2.4525, 2.1547],
        [0.2082, 0.1282]], dtype=torch.float64, requires_grad=True)
```

In [5]:

```python
ghq = GaussHermiteQuadrature(2, 9)
ghq_node_mat = ghq.X.T
```

In [6]:

```python
# Save data as Pytorch tensors.
shares = torch.tensor(np.array(data_ex4['shares']),
                      dtype=torch.double)
covars = torch.tensor(np.array(data_ex4[['const', 'x', 'p']]),
                      dtype=torch.double)
num_covar = covars.size()[1]
instruments = torch.tensor(np.array(data_ex4[['const', 'x', 'z1', 'z2',
                                              'z3', 'z4', 'z5', 'z6']]),
```

```
                                              dtype=torch.double)
x_mat = covars.reshape((num_T, num_prod, num_covar))
s_mat = shares.reshape((num_T, num_prod))

x_random_mat = x_mat[:, :, 1:-1]
```

Part A - Nested Fixed Point Approach

We solve for the model parameters β and $\Gamm$ using the NFXP algorithm outlined in BLP (1995) and Nevo (2001).

In [7]:

```
def mean_utility(b, xi):

    return covars @ b[:, None] + xi

def market_share_val(delta, g):

    # Evaluate the expression for every market, product and
    # Gauss-Hermite node.
    # Returns a matrix of size (num_T, num_prod, GHQ_size).

    numer = torch.exp(delta[:, :, None] + torch.einsum('tjk,kl,lm -> tjm', x_ran
    denom = 1 + numer.sum(axis=1)

    # Compute the share matrix for every value of unobserved individual characte
    share_mat = numer.div(denom[:, None])

    # Take the expected value of the above matrix using a GH integral
    # approximation.
    exp_share = torch.einsum('m, tjm -> tj', ghq.W, share_mat)

    return exp_share

def blp_contraction(b, g, res):

    # Initial guess for mean utility
    delta = mean_utility(b, res).reshape((num_T, num_prod))

    error, tol = 1, 1e-12

    while error > tol:

        exp_delta_new = torch.exp(delta) * s_mat.div(market_share_val(delta, g))

        delta_new = torch.log(exp_delta_new)

        error = torch.linalg.norm(delta_new - delta)
        delta = delta_new

        if error % 20 == 0:
            print('Inner Loop Error = {}'.format(error))

    return delta
```

In [8]:

```
def blp_gmm_loss(b, g):
```

```
    xi = torch.zeros((num_prod * num_T, 1), dtype=torch.double, requires_grad=Fa

    # Obtaining the BLP contraction solution for the mean utilities.
    delta = blp_contraction(b, g, xi).reshape((num_T * num_prod, 1))

    # Run 2SLS of mean utilities on covariates (including prices).
    blp_2sls = IV2SLS(dependent=np.array(delta.detach()),
                      exog=data_ex4[['const', 'x']],
                      endog=data_ex4['p'],
                      instruments=data_ex4[['z1', 'z2', 'z3', 'z4', 'z5', 'z6']]

    # Use 2SLS coefficients.
    b_2sls = torch.tensor(np.array(blp_2sls.params))

    # Derive residuals using 2SLS coefficients.
    res = delta - covars @ b_2sls[:, None]

    # Derive moment conditions required for BLP.
    moment_eqns = res * instruments
    moments = moment_eqns.mean(axis=0)

    loss_gmm = moments[None, :] @ weight_matrix @ moments[:, None]

    print('beta = {}, gamma = {}, loss = {}'.format(np.array(b_2sls.clone().deta
                                                     np.array(g.clone().detach())
                                                     loss_gmm.clone().detach())
          )

    return loss_gmm, moment_eqns, b_2sls
```

In [ ]:

```
opt_gmm = optim.Adam([gamma], lr=0.01)
weight_matrix = Variable(torch.eye(instruments.shape[1], dtype=torch.double), re

# Optimizing over the GMM loss function
for epoch in range(500):

    opt_gmm.zero_grad()   # Reset gradient inside the optimizer

    # Compute the objective at the current parameter values.
    loss, moment_x, new_beta = blp_gmm_loss(beta, gamma)
    loss.backward()    # Gradient computed.

    opt_gmm.step()     # Update parameter values using gradient descent.

    with torch.no_grad():
        gamma[0,1] = gamma[0,1].clamp(0.00, 0.00)
        beta[1] = beta[1].clamp(0.00, np.inf)
        # gamma[0,0] = gamma[0,0].clamp(0.00, np.inf)
        # gamma[1,1] = gamma[1,1].clamp(0.00, np.inf)

    weight_matrix = torch.inverse(1/(num_T * num_prod) * (moment_x.T @ moment_x)
    beta = new_beta.detach()
    # beta = beta2.detach().clone()

    # if epoch % 10 == 0:
    #
    #     loss_val = np.squeeze(loss.detach())
    #     print('Iteration [{}]: Loss = {:2.4e}'.format(epoch, loss_val))
```

In [10]:
```
gamma
```

Out[10]:
```
tensor([[ 2.0276,  0.0000],
        [-0.2168,  0.1702]], dtype=torch.float64, requires_grad=True)
```

In [11]:
```
beta
```

Out[11]:
```
tensor([-4.5439,  0.5183, -0.3859], dtype=torch.float64)
```

In [12]:
```
ω = np.array((gamma @ gamma.T).detach())
ω
```

Out[12]:
```
array([[ 4.1112315 , -0.43952497],
       [-0.43952497,  0.07597359]])
```

In [13]:
```
ω[1,0] / np.sqrt(ω[0,0] * ω[1, 1])
```

Out[13]:
```
-0.7864411888049343
```

In [14]:
```
np.sqrt([ω[0,0] , ω[1,1]])
```

Out[14]:
```
array([2.0276172 , 0.27563308])
```

We find that

$$\widehat{\boldsymbol{\beta}} = \begin{pmatrix} -4.5439 \\ 0.5183 \\ -0.3859 \end{pmatrix}, \qquad \widehat{\boldsymbol{\Gamma}} = \begin{pmatrix} 2.0276 & 0.0000 \\ -0.2168 & 0.1702 \end{pmatrix}$$

In [15]:
```
beta, gamma = beta.detach(), gamma.detach()
```

Part B

To compute market-specific elasticities, we need to first predict individual level market shares for various realizations of $\mathbf{v}_i$ and then average these across all individuals. For each realization of $\mathbf{v}_i$, the predicted market share for product j is given by

$$s_{ijt} = \frac{\exp\left(\widetilde{\mathbf{x}}_{jt}'\boldsymbol{\beta} + \xi_{jt} + \widetilde{\mathbf{x}}_{jt}'\boldsymbol{\Gamma}\boldsymbol{v}_i\right)}{1 + \sum_{k\in\mathcal{J}_t}\exp\left(\widetilde{\mathbf{x}}_{kt}'\boldsymbol{\beta} + \xi_{kt} + \widetilde{\mathbf{x}}_{kt}'\boldsymbol{\Gamma}\boldsymbol{v}_i\right)}$$

The individual coefficients are given by

$$\widehat{\boldsymbol{\beta}}_i = \widehat{\boldsymbol{\beta}} + \boldsymbol{\Gamma}\boldsymbol{v_i}$$

We can put these together to compute the own-price and cross-price elasticities for each market using the following equations:

$$\varepsilon_{jk,t} = \frac{\partial \pi_{j,t}}{\partial p_{k,t}} \frac{p_{k,t}}{\pi_{j,t}} = \begin{cases} -\frac{p_{j,t}}{\pi_{j,t}} \int_{\mathbf{v}_i} \alpha_i \pi_{i,j,t} \left(1 - \pi_{i,j,t}\right) \mathrm{d}F\left(\mathbf{v}_i\right) & \text{if } j = k \\ \frac{p_{k,t}}{\pi_{j,t}} \int_{\mathbf{v}_i} \alpha_i \pi_{i,j,t} \pi_{i,k,t} \,\mathrm{d}F\left(\mathbf{v}_i\right) & \text{otherwise.} \end{cases}$$

We again rely on Gauss-Hermite quadratures to evaluate the integrals.

In [16]:

```
def generate_ind_params(b, g):

    # Predicts individual market shares and individual price coefficients for ea
    # Returns a matrix of size (num_T, num_prod, GHQ_size) and (2, GHQ_size)

    numer = torch.exp(torch.einsum('tjk,kl->tjl', x_mat, b[:, None]) + torch.ein
    denom = 1 + numer.sum(axis=1)

    # Compute the share matrix for every value of unobserved individual characte
    share_mat = numer.div(denom[:, None])

    beta_mat = b[1:-1][:, None] + torch.einsum('kl,lm -> km', g, ghq_node_mat)

    return share_mat, beta_mat
```

In [17]:

```
data_wide = pd.pivot(data_ex4, values='p', index='market', columns='choice')

data_wide.rename(columns={1: "price_1", 2: "price_2",
                          3: "price_3", 4: "price_4",
                          5: "price_5", 6: "price_6"}, inplace=True)

data_ex4 = pd.merge(data_ex4, data_wide, on='market')
```

In [18]:

```
beta[-1] = -beta[-1]

share_mat, beta_mat = generate_ind_params(beta, gamma)

own_price_integral = torch.einsum('tjm, tjm, m, m -> tj', share_mat, 1 - share_m

cross_price_integral = torch.einsum('tjm, tkm, m, m -> tjk', share_mat, share_ma

data_ex4['own'] = - own_price_integral.reshape((num_T * num_prod)) * data_ex4['p

data_ex4['cross_1'] = cross_price_integral[:, :, 0].reshape((num_T * num_prod))
data_ex4['cross_2'] = cross_price_integral[:, :, 1].reshape((num_T * num_prod))
data_ex4['cross_3'] = cross_price_integral[:, :, 2].reshape((num_T * num_prod))
data_ex4['cross_4'] = cross_price_integral[:, :, 3].reshape((num_T * num_prod))
data_ex4['cross_5'] = cross_price_integral[:, :, 4].reshape((num_T * num_prod))
data_ex4['cross_6'] = cross_price_integral[:, :, 5].reshape((num_T * num_prod))
```

In [19]:

```
average_elasticity = data_ex4.groupby('choice')[['own', 'cross_1', 'cross_2', 'c
e_mat = np.array(average_elasticity[['cross_1', 'cross_2', 'cross_3', 'cross_4',
np.fill_diagonal(e_mat, np.array(average_elasticity['own']))
```

j/k	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6
Product 1	-0.00058851	1.01873e-05	0.0383455	0.023619	0.111892	0.103909

j/k	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6
Product 2	2.5978e-05	-0.000799222	0.0416366	0.0259857	0.157375	0.210676
Product 3	0.000215098	0.000272143	-21.8507	4.99784	4.66744	4.99098
Product 4	0.000215119	0.000143814	0.609505	-18.32	4.39132	4.94391
Product 5	6.38342e-05	6.19015e-05	0.273356	0.288736	-5.41066	2.6977
Product 6	4.42949e-05	7.32514e-05	0.215868	0.197864	2.42652	-5.01205

We see that own price and cross price elasticities are not driven solely by functional form, but by the heterogeneity in the price sensitivity across consumers who purchase the various products. This creates the difference between the results here and in Exercise 3. The absurdly low elasticities associated with products 1 and 2 could be driven by the extremely low prices for these products across all markets as seen in the table below for Part 3.

Part 3

The difference in prices and market shares could be attributed to certain products having much lower quality on average (especially products 3 and 4) compared to products 5 and 6. The impact of quality on customer preferences might be heterogenous, but the coefficient related to quality is strictly positive with low variance, which implies that customers will tend to shift away from these products in unison.

In [21]:

```
data_ex4.groupby('choice')[['p', 'x', 'shares']].mean()
```

Out[21]:

choice	p	x	shares
1	0.002439	-0.019330	0.098810
2	0.002286	-0.026036	0.089131
3	2.019113	-0.081252	0.043009
4	1.751616	-0.180135	0.039323
5	3.576978	1.692610	0.151714
6	4.442894	2.002366	0.193238

In [21]:

```

```