ECON 600: Merger Homework

Nicholas Wu

Fall 2021

All code is in Python.

3: Generating Data

(1) See code, boxes [4] and [5].

(2)

(a) (i) We first note that in the parameter specification,

$$\overline{\beta^{(2)}} = 4$$
$$\overline{\beta^{(3)}} = 4$$

Hence, defining, $\sigma^{(2)} = \sigma^{(3)} = 1$, we have that

$$\beta_{it}^{(2)} = \overline{\beta^{(2)}} + \sigma^{(2)} \nu_{it}^{(2)}$$
$$\beta_{it}^{(3)} = \overline{\beta^{(3)}} + \sigma^{(3)} \nu_{it}^{(3)}$$

where $\nu_i^{(2)}$ and $\nu_i^{(3)}$ are i.i.d standard normal.

Define

$$\delta_{jt} = x_{jt} + \overline{\beta^{(2)}} satellite_j + \overline{\beta^{(3)}} wired_j + \alpha p_{jt} + \xi_{jt}$$
$$\mu_{ijt} = \sigma^{(2)} satellite_j \nu_{it}^{(2)} + \sigma^{(3)} wired_j \nu_{it}^{(3)}$$

The multinomial logit choice probabilities are, conditional on all realized coefficients,

$$s_{0t} = \int \frac{1}{Z} \, d\Phi(\nu)$$

$$s_{jt} = \int \frac{\exp(\delta_{jt} + \mu_{ijt})}{Z} \, d\Phi(\nu)$$

for $j > 0$. Where

$$Z = 1 + \sum_{j=1}^{J} \exp(\delta_{jt} + \mu_{ijt})$$

Then the derivatives are

$$\frac{\partial s_{jt}}{\partial p_j} = \int \frac{\alpha \exp(\delta_{jt} + \mu_{ijt}) Z - \exp(\delta_{jt} + \mu_{ijt}) \left(\alpha \exp(\delta_{jt} + \mu_{ijt})\right)}{Z^2} \, d\Phi(\nu)$$

$$\frac{\partial s_{jt}}{\partial p_k} = \int -\frac{\exp(\delta_{jt} + \mu_{ijt}) \left(\alpha \exp(\delta_{kt} + \mu_{ikt})\right)}{Z^2} \, d\Phi(\nu)$$

(ii) See code. The Monte-Carlo simulated derivatives are implemented in block [6].

(iii) See code, block [8]. We were happy with the precision provided by using 3000 draws.

(b) Ok.

(c) (i) See code block [12]. All calls to fsolve converge.

(ii) See code blocks [14], [15]. The maximum difference in any price estimate and any share estimate between the two methods is on the order of 10^{-9} (quite small). We'll just take the fsolve results, since these two seem close anyway.

(3) See code block [17]. Data is placed into the required format.

(4) I am pretty happy with the variation provided by these simulated values. By regressing prices and shares on the within-market observables, we get decent adjusted R^2 values on the regressions on prices (around 0.9) and on shares (around 0.78), as seen in blocks [19], [20]. This suggests that there is enough variation, for my taste at least.

4: Misspecified Models

Table 4.1: Parameter Estimates, Misspecified Models						
Model	α	$\beta^{(1)}$	$\overline{\beta^{(2)}}$	$\overline{\beta^{(3)}}$	$\sigma_{satellite}$	σ_{wired}
OLS Logit	-0.9518	0.8375	1.3705	1.3589		
	(0.044)	(0.029)	(0.122)	(0.123)		
2SLS	-1.8992	0.9461	3.8635	3.8774		
	(0.0700)	(0.0331)	(0.1878)	(0.1880)		
Nested Logit	-1.6649	0.8483	3.4995	3.4881	0.2173	0.1944
	(0.0784)	(0.0377)	(0.1923)	(0.1825)	(0.0770)	(0.0714)

(5) See code block [22]. Parameter estimates are in Table 4.1.

(6) See code block [23]. Parameter estimates are in Table 4.1.

(7) See code block [24]. Parameter estimates are in Table 4.1. Intuitively, the nested logit model is misspecified because the parameters inherently don't allow for the coefficient heterogeneity of the random coefficients model. Even if we allow for group-specific σ parameters, these are merely band-aids to try to fix the substitution patterns (price derivatives of shares) but instrumenting and allowing for these nests do not address the fundamental misspecification of the plain logit model.

(8) We analytically compute the derivatives in the nested logit model. Let

$$\delta_{jt} = \beta^{(1)} x_{jt} + \overline{\beta^{(2)}} satellite_j + \overline{\beta^{(3)}} wired_j + \alpha p_{jt}$$

Then

$$s_{j/g}(\delta_{jt}, \sigma_g) = \frac{\exp(\delta_{jt}/(1-\sigma_g))}{\sum_{i \in \mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))}$$

The own-price derivative is:

$$\frac{\partial}{\partial p_j} s_{j/g}(\delta_{jt}, \sigma_g) = \frac{\left(\sum_{i \in \mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right) \frac{\alpha}{1-\sigma_g} \exp(\delta_{jt}/(1-\sigma_g)) - \frac{\alpha}{1-\sigma_g} \left(\exp(\delta_{jt}/(1-\sigma_g))\right)^2}{\left(\sum_{i \in \mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^2}$$

$$= \frac{\alpha}{1-\sigma_g} s_{j/g}(\delta_{jt}, \sigma_g) \frac{\left(\sum_{i \in \mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right) - \left(\exp(\delta_{jt}/(1-\sigma_g))\right)}{\left(\sum_{i \in \mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)} = \frac{\alpha}{1-\sigma_g} s_{j/g}(\delta_{jt}, \sigma_g)(1 - s_{j/g}(\delta_{jt}, \sigma_g))$$

The within-group price derivative is:

$$\frac{\partial}{\partial p_k} s_{j/g}(\delta_{jt}, \sigma_g) = -\frac{\alpha}{1-\sigma_g} \frac{\exp(\delta_{jt}/(1-\sigma_g))\exp(\delta_{kt}/(1-\sigma_g))}{\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^2}$$

$$= -\frac{\alpha}{1-\sigma_g} s_{j/g}(\delta_t, \sigma_g) s_{k/g}(\delta_t, \sigma_g)$$

The outside-group price derivative of the within-group share is 0. Let δ_t denote the vector of $\delta_j t$ for all j, and let σ denote the vector of σ_g for all g. The group shares are given by

$$s_g(\delta_t, \sigma) = \frac{\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^{1-\sigma_g}}{1+\sum_{g'}\left(\sum_{i\in\mathcal{J}_{g'}} \exp(\delta_{it}/(1-\sigma_{g'}))\right)^{1-\sigma_{g'}}}$$

The within-group price derivative is given by:

$$\frac{\partial}{\partial p_j} s_g(\delta_t, \sigma) = \frac{(1-\sigma_g)\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^{-\sigma_g} \frac{\alpha}{1-\sigma_g}\exp(\delta_{jt}/(1-\sigma_g))}{1+\sum_{g'}\left(\sum_{i\in\mathcal{J}_{g'}} \exp(\delta_{it}/(1-\sigma_{g'}))\right)^{1-\sigma_{g'}}}$$

$$-\frac{\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^{1-\sigma_g}(1-\sigma_g)\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^{-\sigma_g} \frac{\alpha}{1-\sigma_g}\exp(\delta_{jt}/(1-\sigma_g))}{\left(1+\sum_{g'}\left(\sum_{i\in\mathcal{J}_{g'}} \exp(\delta_{it}/(1-\sigma_{g'}))\right)^{1-\sigma_{g'}}\right)^2}$$

$$= \frac{\alpha\left(\sum_{i\in\mathcal{J}_g} \exp(\delta_{it}/(1-\sigma_g))\right)^{-\sigma_g} \exp(\delta_{jt}/(1-\sigma_g))}{1+\sum_{g'}\left(\sum_{i\in\mathcal{J}_{g'}} \exp(\delta_{it}/(1-\sigma_{g'}))\right)^{1-\sigma_{g'}}}(1-s_g(\delta_t,\sigma))$$

$$= \frac{\alpha s_g(\delta_t,\sigma)\exp(\delta_{jt}/(1-\sigma_g))}{\sum_{i\in\mathcal{J}_g}\exp(\delta_{it}/(1-\sigma_g))}(1-s_g(\delta_t,\sigma))$$

$$= \alpha s_g(\delta_t,\sigma) s_{j/g}(\delta_{jt},\sigma_g)(1-s_g(\delta_t,\sigma))$$

$$= \alpha s_j(\delta_t,\sigma)(1-s_g(\delta_t,\sigma))$$

The outside-group price derivative

$$\frac{\partial}{\partial p_k} s_g(\delta_t, \sigma) = -s_g(\delta_t,\sigma)\frac{\alpha\left(\sum_{i\in\mathcal{J}_{g_k}} \exp(\delta_{it}/(1-\sigma_{g_k}))\right)^{-\sigma_{g_k}} \exp(\delta_{kt}/(1-\sigma_{g_k}))}{1+\sum_{g'}\left(\sum_{i\in\mathcal{J}_{g'}} \exp(\delta_{it}/(1-\sigma_{g'}))\right)^{1-\sigma_{g'}}}$$

$$= -s_g(\delta_t,\sigma)\frac{\alpha s_{g_k}(\delta_t,\sigma)\exp(\delta_{kt}/(1-\sigma_{g_k}))}{\sum_{i\in\mathcal{J}_{g_k}}\exp(\delta_{it}/(1-\sigma_{g_k}))}$$

$$= -\alpha s_g(\delta_t,\sigma) s_{g_k}(\delta_t,\sigma) s_{k/g}(\delta_{jt},\sigma_g)$$

$$= -\alpha s_g(\delta_t,\sigma) s_k(\delta_t,\sigma)$$

The market share function is then given by

$$s_j(\delta_t, \sigma) = s_g(\delta_t, \sigma) s_{j/g}(\delta_{jt}, \sigma_g)$$

$$\frac{\partial}{\partial p} s_j(\delta_t, \sigma) = \frac{\partial}{\partial p} s_g(\delta_t, \sigma) s_{j/g}(\delta_{jt}, \sigma_g) + s_g(\delta_t, \sigma) \frac{\partial}{\partial p} s_{j/g}(\delta_{jt}, \sigma_g)$$

The own-price derivative is then:

$$\begin{aligned}
\frac{\partial}{\partial p_j} s_j(\delta_t, \sigma) &= \alpha s_j(\delta_t, \sigma)(1 - s_g(\delta_t, \sigma)) s_{j/g}(\delta_{jt}, \sigma_g) + s_g(\delta_t, \sigma) \frac{\alpha}{1 - \sigma_g} s_{j/g}(\delta_{jt}, \sigma_g)(1 - s_{j/g}(\delta_{jt}, \sigma_g)) \\
&= \alpha s_j(\delta_t, \sigma)(1 - s_g(\delta_t, \sigma)) s_{j/g}(\delta_{jt}, \sigma_g) + s_j(\delta_t, \sigma) \frac{\alpha}{1 - \sigma_g}(1 - s_{j/g}(\delta_{jt}, \sigma_g)) \\
&= \alpha s_j(\delta_t, \sigma) \left((1 - s_g(\delta_t, \sigma)) s_{j/g}(\delta_{jt}, \sigma_g) + \frac{1}{1 - \sigma_g}(1 - s_{j/g}(\delta_{jt}, \sigma_g)) \right) \\
&= \frac{\alpha s_j(\delta_t, \sigma)}{1 - \sigma_g} \left((1 - \sigma_g) s_{j/g}(\delta_{jt}, \sigma_g) - (1 - \sigma_g) s_j(\delta_t, \sigma) + 1 - s_{j/g}(\delta_{jt}, \sigma_g) \right) \\
&= \frac{\alpha s_j(\delta_t, \sigma)}{1 - \sigma_g} \left(1 - \sigma_g s_{j/g}(\delta_{jt}, \sigma_g) - (1 - \sigma_g) s_j(\delta_t, \sigma) \right)
\end{aligned}$$

The within-group price derivative is

$$\begin{aligned}
\frac{\partial}{\partial p_k} s_j(\delta_t, \sigma) &= \alpha s_k(\delta_t, \sigma)(1 - s_g(\delta_t, \sigma)) s_{j/g}(\delta_{jt}, \sigma_g) - s_g(\delta_t, \sigma) \frac{\alpha}{1 - \sigma_g} s_{j/g}(\delta_t, \sigma_g) s_{k/g}(\delta_t, \sigma_g) \\
&= \frac{\alpha}{1 - \sigma_g} s_k(\delta_t, \sigma) \left((1 - \sigma_g) s_{j/g}(\delta_{jt}, \sigma_g) - (1 - \sigma_g) s_j(\delta_t, \sigma) - s_{j/g}(\delta_t, \sigma_g) \right) \\
&= -\frac{\alpha}{1 - \sigma_g} s_k(\delta_t, \sigma) \left(\sigma_g s_{j/g}(\delta_{jt}, \sigma_g) + (1 - \sigma_g) s_j(\delta_t, \sigma) \right)
\end{aligned}$$

The outside-group price derivative is

$$\frac{\partial}{\partial p_k} s_j(\delta_t, \sigma) = -\alpha s_j(\delta_t, \sigma) s_k(\delta_t, \sigma)$$

And the outside-option derivative is

$$\frac{\partial}{\partial p_j} s_0(\delta_t, \sigma) = -\alpha s_0(\delta_t, \sigma) s_j(\delta_t, \sigma)$$

See code blocks [25] and [26] for computation of the derivatives. Results are displayed in blocks [27], [28] and in Table 4.2, 4.3, and 4.4.

Table 4.2: Average Own-Price Elasticities, Nested Logit				
	ε_1	ε_2	ε_3	ε_4
True Values	-4.06535006	-4.16553436	-4.17726162	-4.18978309
Estimated Values	-6.12190672	-6.22209392	-6.06022182	-6.28818366

Table 4.3: True Average Diversion Ratio Matrix			
$\mathcal{D}$: diagonal entries $\mathcal{D}_{jj}$ replaced with $\mathcal{D}_{j0}$			
0.33115087	0.30335128	0.18522023	0.18027762
0.32317153	0.32122579	0.18063565	0.17496703
0.19329289	0.17575241	0.32765373	0.30330097
0.19192008	0.17341037	0.31037504	0.32429451

Table 4.4: Estimated Average Diversion Ratio Matrix, Nested Logit			
$\mathcal{D}$: diagonal entries $\mathcal{D}_{jj}$ replaced with $\mathcal{D}_{j0}$			
0.28479711	0.33297753	0.19134439	0.19088096
0.32541291	0.28728495	0.19641401	0.19088814
0.19547647	0.20791738	0.28870153	0.32231153
0.19719596	0.20488804	0.32444768	0.28792372

5: Estimating the Correctly Specified Model

Code is in blocks [30-34].

(9) See Tables 5.1 and 5.2. We prefer the full model estimation due to the better estimates and more reliable convergence.

(10) Let ε_i denote the own-price elasticity of good i, and let

$$\mathcal{D}_{jk} = -\frac{\partial s_k / \partial p_j}{\partial s_j / \partial p_j}$$

The true and estimated matrix of own-price elasticities is in Table 3, and the true and estimated average diversion ratios are in Table 4 and Table 5, respectively.

Table 5.1: Parameter Estimates, Demand-side Estimation Only					
α	$\beta^{(1)}$	$\overline{\beta^{(2)}}$	$\overline{\beta^{(3)}}$	σ_2	σ_3
-1.852408	0.9872258	3.615042	3.622520	1.0000	1.0000
(0.01867589)	(0.04741592)	(0.04524844)	(0.04691815)	(0.3071605)	(0.3172754)

Table 5.2: Parameter Estimates, Full Model Estimation							
α	$\beta^{(1)}$	$\overline{\beta^{(2)}}$	$\overline{\beta^{(3)}}$	σ_2	σ_3	γ_0	γ_1
-2.0347	1.0568	4.0361	4.0444	1.1782	1.1932	0.49112	0.25381
(0.0858)	(0.0454)	(0.2111)	(0.2131)	(0.2196)	(0.2108)	(0.01772)	(0.00912)

Table 5.3: Average Own-Price Elasticities, Full Model Estimation				
	ε_1	ε_2	ε_3	ε_4
True Values	-4.06535006	-4.16553436	-4.17726162	-4.18978309
Estimated Values	-4.0525488	-4.15853012	-4.16252984	-4.17736646

Table 5.4: True Average Diversion Ratio Matrix			
$\mathcal{D}$: diagonal entries $\mathcal{D}_{jj}$ replaced with $\mathcal{D}_{j0}$			
0.33115087	0.30335128	0.18522023	0.18027762
0.32317153	0.32122579	0.18063565	0.17496703
0.19329289	0.17575241	0.32765373	0.30330097
0.19192008	0.17341037	0.31037504	0.32429451

Table 5.5: Estimated Average Diversion Ratio Matrix, Full Model			
$\mathcal{D}$: diagonal entries $\mathcal{D}_{jj}$ replaced with $\mathcal{D}_{j0}$			
0.32908096	0.32674409	0.1743611	0.16981386
0.34688774	0.31879102	0.16981928	0.16450196
0.18220138	0.16573254	0.32424023	0.32782585
0.18083009	0.16332965	0.33517888	0.32066138

6: Merger Simulation

(11) When two of the firms merge, prices will generically increase for the merged firm's goods. The firms all increase prices because the merged firm can price its own goods closer to monopoly pricing.

(12) See code.

(13) See Table 6.1. Intuitively, it makes sense that the merger of 1 and 2 results in larger price increases than 1 and 3; this is because merging 1 and 2 means the merged firm has a submonopoly on satellite products, and hence has a stronger incentive to raise prices of the satellite TV services.

(14) A reduction in marginal cost means that prices may not necessarily increase as a result of the merger, and hence can potentially improve efficiency; the merged firm can earn more profits to outweigh any consumer welfare decrease. If the marginal cost decrease is very large, it is even possible for consumer welfare to also increase.

(15) Code is in blocks [46-52]. See Table 6.1 for the post-merger prices with cost reduction. The net consumer welfare actually decreases as a result of the merger by 6.8384. However, the firm manages to earn significantly more profits: specifically, the firm earns 69.3230 more in profits. Hence the overall predicted welfare change is 62.4846. We need to assume the markets have uniform measure of consumers here because previously all the computations were performed using in-market shares, which has no reliance on the size of the market. For net consumer welfare and profits, we have to aggregate across markets, and hence we need assumptions on the measure of consumers in each market.

Table 6.1: Average Prices across Markets, Merger Analysis				
	p_1	p_2	p_3	p_4
Pre-Merger	2.7327	2.7165	2.7608	2.7391
Merging 1 and 2	2.9808	2.9949	2.7712	2.7488
Merging 1 and 3	2.8464	2.7285	2.8826	2.7514
Merging 1 and 2, with cost decrease	2.7833	2.7954	2.7612	2.7391

Appendix: Code

[1]:
```
import numpy as np
from scipy.optimize import fsolve, fixed_point
from matplotlib import pyplot as plt
import pyblp
from tqdm.notebook import trange
import statsmodels.api as sm
import statsmodels.formula.api as smf
from linearmodels.iv import IV2SLS
import pandas as pd
```

[2]:
```
RNG_SEED = 8476263

rng = np.random.default_rng(RNG_SEED) # this random seeding is for reproducibility
```

[3]:
```
# I am horrified that we have to overrun the default collinearity checks
# however, wired and satellite dummy variables are collinear
# so to prevent PyBLP from throwing a fit, we must do this.
pyblp.options.collinear_rtol = 0
pyblp.options.collinear_atol = 0
```

[4]:
```
# fixed parameter definitions

beta1 = 1
alpha = -2
gamma0 = 1/2
gamma1 = 1/4
beta2_bar = 4
beta3_bar = 4
sigma2 = 1
sigma3 = 1

# markets and goods
T = 600
J = 4
```

[5]:
```
# 3.1

# x_jt, w_jt are absolute value of iid standard normal variables
x = np.absolute(rng.standard_normal(size=(J,T)))
```

```python
w = np.absolute(rng.standard_normal(size=(J,T)))

unobservable_mean = [0,0]
unobservable_cov = [[1,0.25],[0.25,1]]
unobservables = rng.multivariate_normal(unobservable_mean, unobservable_cov,␣
 ↪size=(J,T))
xi = unobservables[:,:,0]
omega = unobservables[:,:,1]
```

[6]:
```python
# 3.2a
# defining the market share

def own_mkt_share_derivative(t, p, beta2, beta3):
    # p should be a length J vector
    # betas should be num_sims

    u_t = np.tile(x[:,t] + xi[:,t] + alpha*p, (len(beta2), 1)) # num_sims x J
    for j in range(J):
        if j < 2:
            u_t[:,j] = u_t[:,j] + beta2
        else:
            u_t[:,j] = u_t[:,j] + beta3

    Z = np.tile( 1 + np.sum(np.exp(u_t),axis=-1), (J,1)).T
    numerator = alpha*np.exp(u_t)*Z - alpha*np.square(np.exp(u_t)) # num_sims x J
    denominator = np.square(Z)

    return np.mean(numerator / denominator, axis=0)

def outside_mkt_share_derivative(t, p, beta2, beta3):
    # p should be a length J vector
    # betas should be num_sims

    u_t = np.tile(x[:,t] + xi[:,t] + alpha*p, (len(beta2), 1)) # num_sims x J
    for j in range(J):
        if j < 2:
            u_t[:,j] = u_t[:,j] + beta2
        else:
            u_t[:,j] = u_t[:,j] + beta3

    Z = np.tile( 1 + np.sum(np.exp(u_t),axis=-1), (J,1)).T
```

```python
    numerator = -1*alpha*np.exp(u_t) # num_sims x J
    denominator = np.square(Z)

    return np.mean(numerator / denominator, axis=0)

def full_mkt_share_derivative(t, p, beta2, beta3):
    # p should be a length J vector
    # betas should be num_sims

    u_t = np.tile(x[:,t] + xi[:,t] + alpha*p, (len(beta2), 1)) # num_sims x J
    for j in range(J):
        if j < 2:
            u_t[:,j] = u_t[:,j] + beta2
        else:
            u_t[:,j] = u_t[:,j] + beta3

    Z = np.tile( 1 + np.sum(np.exp(u_t),axis=-1), (J,1)).T # num_sims x J

    derivatives = np.zeros((J,J))

    own_numerator = alpha*np.exp(u_t)*Z - alpha*np.square(np.exp(u_t)) # num_sims x J
    denominator = np.square(Z)

    for j in range(J):
        derivatives[j,j] = np.mean(own_numerator / denominator, axis=0)[j]

    for j in range(J):
        for k in range(J):
            if not (j == k):
                derivatives[j,k] = np.mean(-1*alpha*np.exp(u_t)[:,k]*np.exp(u_t)[:,j] /
 ↪ np.square(1 + np.sum(np.exp(u_t),axis=-1)))

    return derivatives
```

[7]:
```python
# s_jt(p)
def mkt_share(t, p, beta2, beta3):
    # p should be a length J vector
    # betas should be num_sims

    u_t = np.tile(x[:,t] + xi[:,t] + alpha*p, (len(beta2), 1)) # num_sims x J
    for j in range(J):
```

```
        if j < 2:
            u_t[:,j] = u_t[:,j] + beta2
        else:
            u_t[:,j] = u_t[:,j] + beta3

    numerator = np.exp(u_t)
    denominator = 1 + np.sum(np.exp(u_t),axis=-1) # num_sims

    return np.mean(numerator / (np.tile(denominator, (J, 1)).T), axis=0)
```

[8]:
```
# 3.2a(iv)
# draw beta coefficients for N individuals S times, observe variation in market share␣
↪derivatives

S = 100

all_derivatives = np.zeros((J,J,S))
all_shares = np.zeros((J,S))

N = 3000

price = np.array([1,1,1,1])

for s in trange(S):
    beta2 = rng.normal(beta2_bar, sigma2, N)
    beta3 = rng.normal(beta3_bar, sigma3, N)
    all_derivatives[:,:,s] = full_mkt_share_derivative(0, price, beta2, beta3)
    all_shares[:,s] = mkt_share(1, price, beta2, beta3)
(np.mean(all_shares,axis=1), np.std(all_shares,axis=1), np.mean(all_derivatives,␣
↪axis=2), np.std(all_derivatives, axis=2))
```

```
HBox(children=(IntProgress(value=0), HTML(value='')))
```

[8]:
```
(array([0.04784253, 0.15288083, 0.44046486, 0.35277411]),
 array([0.0008991 , 0.00287306, 0.00211771, 0.0016961 ]),
 array([[-0.29478105,  0.06650959,  0.2168944 ,  0.00709914],
        [ 0.06650959, -0.16315672,  0.09183023,  0.00300568],
        [ 0.2168944 ,  0.09183023, -0.35012373,  0.03100105],
        [ 0.00709914,  0.00300568,  0.03100105, -0.04144621]]),
```

```
array([[3.08401066e-03, 1.64034900e-03, 1.89106999e-03, 6.18963701e-05],
       [1.64034900e-03, 2.14278030e-03, 8.00654110e-04, 2.62061074e-05],
       [1.89106999e-03, 8.00654110e-04, 2.45783477e-03, 3.51894072e-04],
       [6.18963701e-05, 2.62061074e-05, 3.51894072e-04, 2.89493485e-04]]))
```

[9]:
```python
mc = np.exp( gamma0 + gamma1*w + omega/8)
```

[10]:
```python
# define function to solve

def get_function_to_solve(t, beta2, beta3):
    def F(p):
        # p is a
        ds_dp = own_mkt_share_derivative(t, p, beta2, beta3)
        shares = mkt_share(t, p, beta2, beta3)
        return p - mc[:,t] + np.reciprocal(ds_dp)*shares

    return F
```

[11]:
```python
# draw betas, now compute equilibrium prices and shares

beta2 = rng.normal(beta2_bar, sigma2, (N,T))
beta3 = rng.normal(beta3_bar, sigma3, (N,T))
```

[12]:
```python
# 3.2 and 3.3: compute equilibrium shares, prices

# these two variables are the prices and shares
eq_prices = np.zeros((J, T))
eq_shares = np.zeros((J, T))

flag_total = 0

for t in trange(T):
    fn = get_function_to_solve(t, beta2[:,t], beta3[:,t])
    mkt_eq_prices, _ , flag, _ = fsolve(fn, np.array([1,1,1,1]), full_output=True)
    flag_total += flag
    eq_prices[:,t] = mkt_eq_prices
    eq_shares[:, t] = mkt_share(t, mkt_eq_prices, beta2[:,t], beta3[:,t])

# this should be True iff all of the fsolves converge
flag_total == T
```

```
HBox(children=(IntProgress(value=0, max=600), HTML(value='')))
```

[12]: True

[13]:
```
# check that at the equilibrium prices, the estimates for market shares and market␣
↪share derivatives are precise
# repeating the exercise of simulation with equilibrium prices, trying to get␣
↪equilibrium shares

S = 100

all_derivatives = np.zeros((J,J,S))
all_shares = np.zeros((J,S))

N = 100

for t in trange(T):
    price = np.array(eq_prices[:,t])
    for s in range(S):
        beta2_s = np.random.normal(beta2_bar, sigma2, N)
        beta3_s = np.random.normal(beta3_bar, sigma3, N)
        all_derivatives[:,:,s] = full_mkt_share_derivative(0, price, beta2_s, beta3_s)
        all_shares[:,s] = mkt_share(t, price, beta2_s, beta3_s)

(np.mean(all_shares,axis=1), np.std(all_shares,axis=1), np.mean(all_derivatives,␣
↪axis=2), np.std(all_derivatives, axis=2))
```

```
HBox(children=(IntProgress(value=0, max=600), HTML(value='')))
```

[13]:
```
(array([0.25025446, 0.1106518 , 0.40782337, 0.08575528]),
 array([0.01739545, 0.00769152, 0.02222747, 0.00467389]),
 array([[-0.3235269 ,  0.04609141,  0.19011712,  0.00800107],
        [ 0.04609141, -0.1104657 ,  0.04411352,  0.00185652],
        [ 0.19011712,  0.04411352, -0.39876114,  0.02550838],
        [ 0.00800107,  0.00185652,  0.02550838, -0.0412167 ]]),
 array([[1.66922701e-02, 5.70831379e-03, 9.55360361e-03, 4.02062959e-04],
        [5.70831379e-03, 7.73063888e-03, 2.21675482e-03, 9.32920224e-05],
        [9.55360361e-03, 2.21675482e-03, 1.02488422e-02, 2.18776846e-03],
```

```
[4.02062959e-04, 9.32920224e-05, 2.18776846e-03, 2.04859466e-03]]))
```

[14]:
```
# Morrow and Skerlos (2011) Method: (see equation 27 in Conlon + Gortmaker)

def get_matrices(t, p, beta2, beta3):
    # p should be a length J vector
    # betas should be num_sims

    u_t = np.tile(x[:,t] + xi[:,t] + alpha*p, (len(beta2), 1)) # num_sims x J
    for j in range(J):
        if j < 2:
            u_t[:,j] = u_t[:,j] + beta2
        else:
            u_t[:,j] = u_t[:,j] + beta3

    Z = np.tile( 1 + np.sum(np.exp(u_t),axis=-1), (J,1)).T # num_sims x J

    Lambda_inv = np.zeros((J,J))
    Gamma = np.zeros((J,J))

    own_numerator = alpha*np.exp(u_t)  # num_sims x J
    denominator = Z

    for j in range(J):
        Lambda_inv[j,j] = 1 / (np.mean(own_numerator / denominator, axis=0)[j])

    for j in range(J):
        for k in range(J):
            Gamma[j,k] = np.mean(alpha*np.exp(u_t)[:,k]*np.exp(u_t)[:,j] / np.square(1␣
↪+ np.sum(np.exp(u_t),axis=-1)))

    return Lambda_inv, Gamma

def get_fixed_point_function(t, beta2, beta3):
    ownership_matrix = np.identity(J)
    def F(p):
        Lambda_inv, Gamma = get_matrices(t, p, beta2, beta3)
        shares = mkt_share(t, p, beta2, beta3)
        zeta = np.matmul(np.matmul(Lambda_inv, ownership_matrix*Gamma), (p - mc[:,t]))␣
↪- np.matmul(Lambda_inv, shares)
        return mc[:,t] + zeta
```

```
    return F
```

[15]:
```
# Simulate equilibrium using the Morrow and Skerlos (2011) method
eq_prices_2 = np.zeros((J, T))
eq_shares_2 = np.zeros((J, T))

for t in trange(T):
    fn = get_fixed_point_function(t, beta2[:,t], beta3[:,t])
    mkt_eq_prices = fixed_point(fn, np.array([1,1,1,1]), method="iteration")
    eq_prices_2[:,t] = mkt_eq_prices
    eq_shares_2[:, t] = mkt_share(t, mkt_eq_prices, beta2[:,t], beta3[:,t])

# the difference between the two methods, check that this is small
np.max(eq_prices_2 - eq_prices), np.max(eq_shares_2 - eq_shares)
```

```
HBox(children=(IntProgress(value=0, max=600), HTML(value='')))
```

[15]:
```
(1.373072322508051e-09, 4.207107107134789e-09)
```

[16]:
```
# Precompute the price elasticities and diversion
# What PyBLP does, and what we will do, is replace the diagonal of the diversion ratio␣
 ↪matrix with the outside option diversion ratio (instead of -1)
true_price_elasticities = np.zeros((J,J,T))
true_diversion_ratios = np.zeros((J,J,T))

N = 100

for t in trange(T):
    own_price_derivative = own_mkt_share_derivative(t, eq_prices[:,t], beta2[:,t],␣
 ↪beta3[:,t])
    derivative_matrix = full_mkt_share_derivative(t, eq_prices[:,t], beta2[:,t],␣
 ↪beta3[:,t])
    true_price_elasticities[:,:,t] = eq_prices[:,t]*derivative_matrix / eq_shares[:,t].
 ↪T
    derivative_matrix = full_mkt_share_derivative(t, eq_prices[:,t], beta2[:,t],␣
 ↪beta3[:,t])
    outside_derivatives = outside_mkt_share_derivative(t, eq_prices[:,t], beta2[:,t],␣
 ↪beta3[:,t])
```

```
    for j in range(J):
        for k in range(J):
            true_diversion_ratios[j,k,t] = -1*derivative_matrix[k,j]/
↪derivative_matrix[j,j]
    for j in range(J):
        true_diversion_ratios[j,j,t] = -1*outside_derivatives[j]/derivative_matrix[j,j]
```

```
HBox(children=(IntProgress(value=0, max=600), HTML(value='')))
```

[17]:
```
market_ids = np.tile(np.arange(T) + 1,(J,1)).T.flatten()
firm_ids = np.tile(np.arange(J) + 1,(T,1)).flatten()
satellite = np.concatenate((np.ones((2,T)), np.zeros((2,T)))).T.flatten()
wired = np.concatenate((np.zeros((2,T)), np.ones((2,T)))).T.flatten()
observed_data = pd.DataFrame(data={
    "market_ids": market_ids,
    "firm_ids": firm_ids,
    "shares": eq_shares.T.flatten(),
    "prices": eq_prices.T.flatten(),
    "x": x.T.flatten(),
    "satellite": satellite,
    "wired": wired,
    "w": w.T.flatten()
})
unobserved_data = pd.DataFrame(data={
    "market_ids": market_ids,
    "firm_ids": firm_ids,
    "xi": xi.T.flatten(),
    "omega": omega.T.flatten()
})
```

[18]:
```
# Instrument Analysis

df1 = pd.DataFrame({
    'p1':eq_prices[0,:],
    "s1":eq_shares[0,:],
    'p2':eq_prices[1,:],
    "s2":eq_shares[1,:],
    'p3':eq_prices[2,:],
    "s3":eq_shares[2,:],
```

```python
    'p4':eq_prices[3,:],
    "s4":eq_shares[3,:],
    'x1':pd.Series(x[0,:]),
    'w1':pd.Series(w[0,:]),
    'x2':pd.Series(x[1,:]),
    'w2':pd.Series(w[1,:]),
    'x3':pd.Series(x[2,:]),
    'w3':pd.Series(w[2,:]),
    'x4':pd.Series(x[3,:]),
    'w4':pd.Series(w[3,:]),

})

X = df1[["x1","x2","x3","x4","w1","w2","w3","w4"]]
# regress prices on observables

modelp1 = sm.OLS(df1["p1"],X).fit()
modelp2 = sm.OLS(df1["p2"],X).fit()
modelp3 = sm.OLS(df1["p3"],X).fit()
modelp4 = sm.OLS(df1["p4"],X).fit()
models1 = sm.OLS(df1["s1"],X).fit()
models2 = sm.OLS(df1["s2"],X).fit()
models3 = sm.OLS(df1["s3"],X).fit()
models4 = sm.OLS(df1["s4"],X).fit()
```

[19]:
```python
modelp1.rsquared_adj, modelp2.rsquared_adj, modelp3.rsquared_adj, modelp4.rsquared_adj
```

[19]:
```
(0.9435061522877474,
 0.9480775331245905,
 0.9468682020346536,
 0.9477482549660268)
```

[20]:
```python
models1.rsquared_adj, models2.rsquared_adj, models3.rsquared_adj, models4.rsquared_adj
```

[20]:
```
(0.7644818350345643,
 0.7983742915490801,
 0.7648735858119549,
 0.7737755052449837)
```

0.1 Part 4

[21]:
```
model_data = observed_data.copy()
model_data["x_other"] = np.stack([
    x[1,:]+x[2,:]+x[3,:],
    x[0,:]+x[2,:]+x[3,:],
    x[0,:]+x[1,:]+x[3,:],
    x[0,:]+x[1,:]+x[2,:]]).T.flatten()
model_data["w_other"] = np.stack(
    [w[1,:]+w[2,:]+w[3,:],
     w[0,:]+w[2,:]+w[3,:],
     w[0,:]+w[1,:]+w[3,:],
     w[0,:]+w[1,:]+w[2,:]]).T.flatten()
```

[22]:
```
# 4A: Logit
outside_shares = 1 - np.sum(eq_shares, axis=0, keepdims=True)
y = np.log(eq_shares/outside_shares).T.flatten()
X = model_data[["x","satellite","wired","prices"]]
results = sm.OLS(y,X).fit()
results.summary()
```

[22]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.314
Model:                            OLS   Adj. R-squared:                  0.313
Method:                 Least Squares   F-statistic:                     365.4
Date:                Wed, 13 Oct 2021   Prob (F-statistic):          2.09e-195
Time:                        21:30:31   Log-Likelihood:                -3033.1
No. Observations:                2400   AIC:                             6074.
Df Residuals:                    2396   BIC:                             6097.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x              0.8375      0.029     28.572      0.000       0.780       0.895
satellite      1.3705      0.122     11.239      0.000       1.131       1.610
wired          1.3589      0.123     11.046      0.000       1.118       1.600
prices        -0.9518      0.044    -21.393      0.000      -1.039      -0.865
```

```
==============================================================================
Omnibus:                       41.828   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               48.815
Skew:                          -0.263   Prob(JB):                     2.51e-11
Kurtosis:                       3.460   Cond. No.                         30.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

Note that ignoring the endogeneity of prices results in underestimating the magnitudes of all the relevant parameters.

[23]:
```
#6: IV-2SLS
X_exog = model_data[["x", "satellite", "wired"]].astype(float)
X_endog = model_data[["prices"]].astype(float)
Z = model_data[["w", "x_other", "w_other"]].astype(float)
# we'll just instrument for prices with w and x of the own-product; it seems good␣
 ↪enough here
iv_model = IV2SLS(y, X_exog, X_endog, Z).fit()
iv_model.summary
```

[23]:
```
<class 'linearmodels.compat.statsmodels.Summary'>
"""
                          IV-2SLS Estimation Summary
==============================================================================
Dep. Variable:              dependent   R-squared:                      0.1841
Estimator:                    IV-2SLS   Adj. R-squared:                 0.1831
No. Observations:                2400   F-statistic:                    2007.7
Date:                Wed, Oct 13 2021   P-value (F-stat)                0.0000
Time:                        21:30:31   Distribution:                  chi2(4)
Cov. Estimator:                robust

                             Parameter Estimates
==============================================================================
            Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
------------------------------------------------------------------------------
x              0.9461     0.0331     28.609     0.0000      0.8813      1.0109
satellite      3.8635     0.1878     20.575     0.0000      3.4955      4.2315
```

```
wired           3.8774     0.1880     20.628     0.0000     3.5090     4.2458
prices         -1.8992     0.0700    -27.132     0.0000    -2.0364    -1.7620
==============================================================================

Endogenous: prices
Instruments: w, x_other, w_other
Robust Covariance (Heteroskedastic)
Debiased: False
"""
```

[24]:

```
#4.7 nested logit

# construct log of within group share
satellite_share= eq_shares[0,:] + eq_shares[1,:]
wired_share= eq_shares[2,:] + eq_shares[3,:]
model_data["within_satellite_shares"] = model_data["satellite"]*np.log(eq_shares /␣
 ↪satellite_share).T.flatten()
model_data["within_wired_shares"] = model_data["wired"]*np.log(eq_shares /␣
 ↪wired_share).T.flatten()
model_data["within_group_shares"] = model_data["within_wired_shares"] +␣
 ↪model_data["within_satellite_shares"]

# now use the other in-group firm's characteristics as instruments
model_data["x_other_satellite"] = np.stack([x[1,:], x[0,:], x[3,:], x[2,:]]).T.
 ↪flatten()*model_data["satellite"]
model_data["w_other_satellite"] = np.stack([w[1,:], w[0,:], w[3,:], w[2,:]]).T.
 ↪flatten()*model_data["satellite"]
model_data["x_other_wired"] = np.stack([x[1,:], x[0,:], x[3,:], x[2,:]]).T.
 ↪flatten()*model_data["wired"]
model_data["w_other_wired"] = np.stack([w[1,:], w[0,:], w[3,:], w[2,:]]).T.
 ↪flatten()*model_data["wired"]

X_exog = model_data[["x","satellite","wired"]]
X_endog = model_data[["prices", "within_satellite_shares", "within_wired_shares" ]]
Z = model_data[["w", "x_other", "w_other", "x_other_satellite", "w_other_satellite",␣
 ↪"x_other_wired", "w_other_wired"]]
iv_model = IV2SLS(y, X_exog, X_endog, Z).fit()
iv_model.summary
```

[24]: <class 'linearmodels.compat.statsmodels.Summary'>

```
"""
                          IV-2SLS Estimation Summary
==============================================================================
Dep. Variable:              dependent   R-squared:                      0.3589
Estimator:                    IV-2SLS   Adj. R-squared:                 0.3576
No. Observations:                2400   F-statistic:                    2635.4
Date:                Wed, Oct 13 2021   P-value (F-stat)                0.0000
Time:                        21:30:31   Distribution:                  chi2(6)
Cov. Estimator:                robust

                                   Parameter Estimates
================================================================================
===========
                         Parameter  Std. Err.     T-stat    P-value    Lower CI
Upper CI
--------------------------------------------------------------------------------
-----------
x                           0.8483     0.0377     22.479     0.0000      0.7743
0.9223
satellite                   3.4995     0.1923     18.200     0.0000      3.1226
3.8763
wired                       3.4881     0.1825     19.111     0.0000      3.1303
3.8458
prices                     -1.6649     0.0784    -21.241     0.0000     -1.8185
-1.5112
within_satellite_shares     0.2173     0.0770     2.8237     0.0047      0.0665
0.3681
within_wired_shares         0.1944     0.0714     2.7237     0.0065      0.0545
0.3342
================================================================================
===========

Endogenous: prices, within_satellite_shares, within_wired_shares
Instruments: w, x_other, w_other, x_other_satellite, w_other_satellite,
x_other_wired, w_other_wired
Robust Covariance (Heteroskedastic)
Debiased: False
"""
```

[25]:
```
# define functions for derivatives and shares in the nested logit

def full_mkt_share_derivative_nested(t, p, pars):

    XX =  [x[:,t], [1,1,0,0], [0,0,1,1], p  ]

    v_t = pars[0:4]  @ XX

    sigma_1 = pars[4]

    sigma_2 = pars[5]

    theta_2 = pars[3]


    D1 = np.exp(v_t[0]/(1- sigma_1)) + np.exp(v_t[1]/(1- sigma_1))

    D2 = np.exp(v_t[2]/(1- sigma_2)) + np.exp(v_t[3]/(1- sigma_2))

    Z = 1 + np.power(D1, (1- sigma_1)) + np.power(D2, (1- sigma_2))


    derivatives = np.zeros((J,J))

    for j in range(J):
        if j < 2:
            derivatives[j,j] = (theta_2/(1 - sigma_1))*( np.exp(v_t[j]/(1-␣
↪sigma_1))*np.power(D1, sigma_1)*Z -  np.exp((2*v_t[j])/(1- sigma_1))*(   sigma_1*np.
↪power(D1, sigma_1 -1)*Z + (1- sigma_1)) )  / np.square((np.power(D1, sigma_1)*Z))
        else:
            derivatives[j,j] = (theta_2/(1 - sigma_2))*( np.exp(v_t[j]/(1-␣
↪sigma_2))*np.power(D2, sigma_2)*Z -  np.exp((2*v_t[j])/(1- sigma_2))*(   sigma_2*np.
↪power(D2, sigma_2 -1)*Z + (1- sigma_2)) )  / np.square((np.power(D2, sigma_2)*Z))


    for j in range(J):
        for k in range(J):
            if not (j == k):
                if j < 2 and k < 2:
```

```
                    derivatives[j,k] = (-theta_2/(1 - sigma_1))*np.exp(v_t[j]/(1-␣
↪sigma_1))*np.exp(v_t[k]/(1- sigma_1))*( sigma_1*np.power(D1,sigma_1-1)*Z + (1-␣
↪sigma_1)   )  / np.square((np.power(D1, sigma_1)*Z))
                if j >= 2 and k >= 2:
                    derivatives[j,k] = (-theta_2/(1 - sigma_2))*np.exp(v_t[j]/(1-␣
↪sigma_2))*np.exp(v_t[k]/(1- sigma_2))*( sigma_1*np.power(D2,sigma_2-1)*Z + (1-␣
↪sigma_2)   )  / np.square((np.power(D2, sigma_2)*Z))
                if j < 2 and k >= 2:
                    derivatives[j,k] = (-theta_2/(1 - sigma_2))*np.exp(v_t[j]/(1-␣
↪sigma_1))*np.exp(v_t[k]/(1- sigma_2))*(1-sigma_2)*np.power(D2, - sigma_2)*np.
↪power(D1, sigma_1) / np.square((np.power(D1, sigma_1)*Z))
                if j >= 2 and k < 2:
                    derivatives[j,k] = (-theta_2/(1 - sigma_1))*np.exp(v_t[j]/(1-␣
↪sigma_2))*np.exp(v_t[k]/(1- sigma_1))*(1-sigma_1)*np.power(D1, - sigma_1)*np.
↪power(D2, sigma_2) / np.square((np.power(D2, sigma_2)*Z))


    estimated_shares = mkt_share_nested(t, p, pars)
    return derivatives, -1*theta_2*estimated_shares*(1 - estimated_shares.sum())

def mkt_share_nested(t, p, pars):

    XX =  [x[:,t], [1,1,0,0], [0,0,1,1], p  ]

    v_t = pars[0:4] @XX

    sigma_1 = pars[4]

    sigma_2 = pars[5]

    theta_2 = pars[3]


    D1 = np.exp(v_t[0]/(1- sigma_1)) + np.exp(v_t[1]/(1- sigma_1))

    D2 = np.exp(v_t[2]/(1- sigma_2)) + np.exp(v_t[3]/(1- sigma_2))

    Z = 1 + np.power(D1, (1- sigma_1)) + np.power(D2, (1- sigma_2))
```

```
    shares = np.zeros((J,1))

    for j in range(J):
        if j < 2:
            shares[j] = (np.exp(v_t[j]/(1-sigma_1))) /  (np.power(D1, sigma_1)*Z)
        else:
            shares[j] = (np.exp(v_t[j]/(1-sigma_2))) /  (np.power(D2, sigma_2)*Z)


    return shares
```

[26]:
```
# Precompute the price elasticities and diversion
nested_logit_price_elasticities = np.zeros((J,J,T))
nested_logit_diversion_ratios = np.zeros((J,J,T))

N = 100

for t in trange(T):
    derivative_matrix, outside_derivative = full_mkt_share_derivative_nested(t,␣
 ↪eq_prices[:,t], iv_model.params)
    nested_logit_price_elasticities[:,:,t] = eq_prices[:,t]*derivative_matrix /␣
 ↪eq_shares[:,t].T
    estimated_shares = mkt_share_nested(t, eq_prices[:,t], iv_model.params)
    for j in range(J):
        for k in range(J):
            nested_logit_diversion_ratios[j,k,t] = -1*derivative_matrix[k,j]/
 ↪derivative_matrix[j,j]
        nested_logit_diversion_ratios[j,j,t] = -1*outside_derivative[j]/
 ↪derivative_matrix[j,j]
```

```
HBox(children=(IntProgress(value=0, max=600), HTML(value='')))
```

[27]:
```
nested_logit_price_elasticities.mean(axis=2), true_price_elasticities.mean(axis=2)
```

[27]:
```
(array([[-6.12190672,  1.99361387,  1.14721285,  1.16871821],
        [ 1.94980523, -6.22209392,  1.23343822,  1.28434919],
        [ 1.18472154,  1.16184628, -6.06022182,  2.04574024],
        [ 1.15410091,  1.21673082,  1.99322672, -6.28818366]]),
```

```
 array([[-4.06535006,  1.38543391,  0.80172334,  0.7895892 ],
        [ 1.27934133, -4.16553436,  0.71112989,  0.71512854],
        [ 0.73928313,  0.74163481, -4.17726162,  1.3416553 ],
        [ 0.72070405,  0.7189693 ,  1.30923805, -4.18978309]]))
```

[28]:
```python
nested_logit_diversion_ratios.mean(axis=2), true_diversion_ratios.mean(axis=2)
```

[28]:
```
(array([[0.28479711, 0.33297753, 0.19134439, 0.19088096],
        [0.32541291, 0.28728495, 0.19641401, 0.19088814],
        [0.19547647, 0.20791738, 0.28870153, 0.32231153],
        [0.19719596, 0.20488804, 0.32444768, 0.28792372]]),
 array([[0.33115087, 0.30335128, 0.18522023, 0.18027762],
        [0.32317153, 0.32122579, 0.18063565, 0.17496703],
        [0.19329289, 0.17575241, 0.32765373, 0.30330097],
        [0.19192008, 0.17341037, 0.31037504, 0.32429451]]))
```

1 Part 5

1.1 5.a: Demand-side Estimation only

[29]:
```python
# BLP, Demand-side estimation only

demand_problem = pyblp.Problem(
    [
        pyblp.Formulation("0 + prices + x + satellite + wired"),
        pyblp.Formulation("0 + satellite + wired")
    ],
    observed_data[[
        "market_ids",
        "firm_ids",
        "shares",
        "prices",
        "x",
        "satellite",
        "wired"]],
    integration=pyblp.Integration('product', size=9),
)
```

```
Initializing the problem ...
Initialized the problem after 00:00:00.
```

```
Dimensions:
=======================================
 T    N     F     I     K1    K2    MD
---  ----  ---  -----  ----  ----  ----
600  2400   4   48600   4     2     3
=======================================

Formulations:
=================================================================
       Column Indices:            0        1        2        3
----------------------------  ---------  -----  ---------  -----
 X1: Linear Characteristics     prices     x    satellite  wired
X2: Nonlinear Characteristics  satellite  wired
=================================================================
```

[30]:

```python
# we will assume that the random coefficients on satellite and wired are uncorrellated

# this step is going to spit out a lot of text, most of which is not meaningful yet.
# the first iteration of .solve is only to compute the optimal instruments, and hence␣
↪these first-step estimates are not very good

demand_problem_w_instruments = demand_problem.solve(sigma=np.identity(2)).
↪compute_optimal_instruments().to_problem()
```

```
Solving the problem ...

Nonlinear Coefficient Initial Values:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +1.000000E+00
  wired    +0.000000E+00  +1.000000E+00
=======================================

Nonlinear Coefficient Lower Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +0.000000E+00
  wired    +0.000000E+00  +0.000000E+00
=======================================
```

```
Nonlinear Coefficient Upper Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite      +INF
  wired    +0.000000E+00      +INF
=======================================

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm             Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  ----------------------------
 1         0             1          4478         13776        0
+1.397861E-27                 +1.276800E-13  +1.000000E+00, +1.000000E+00

Optimization completed after 00:00:02.
Computing the Hessian and and updating the weighting matrix ...
Computed results after 00:00:10.

Problem Results Summary:
===============================================================================
===============
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------
 1    +1.397861E-27  +1.276800E-13   +1.507973E-06    +7.409371E-06      0
+1.043996E+01
===============================================================================
===============

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
```

```
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm             Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  ----------------------------
 2         0             1           0           600         0
+1.073274E-27                 +3.029540E-13  +1.000000E+00, +1.000000E+00

Optimization completed after 00:00:01.
Computing the Hessian and estimating standard errors ...
Computed results after 00:00:07.

Problem Results Summary:
================================================================================
==================================
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix  Covariance Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number  Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------  -----------------
 2    +1.073274E-27  +3.029540E-13   -2.208656E-05    +8.266331E-06      0
+9.227456E+00      +4.366231E+17
================================================================================
==================================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:20       Yes          0             4          4478         14376
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
===========================================
 Sigma:       satellite          wired
---------  ---------------  ---------------
satellite   +1.000000E+00
           (+4.410576E-03)
```

```
  wired     +0.000000E+00    +1.000000E+00
                            (+4.532765E-03)
===========================================

Beta Estimates (Robust SEs in Parentheses):
==================================================================
    prices              x           satellite         wired
---------------  ---------------  ---------------  ---------------
 -4.507325E-01    +8.461436E-01    -8.777418E-02    -1.191342E-01
(+1.126041E-02)  (+3.082421E-02)  (+1.866351E-02)  (+1.861941E-02)
==================================================================
Computing optimal instruments for theta ...
Computed optimal instruments after 00:00:01.

Optimal Instrument Results Summary:
=======================
Computation  Error Term
   Time        Draws
-----------  ----------
 00:00:01        1
=======================
Re-creating the problem ...
Re-created the problem after 00:00:00.

Dimensions:
=======================================
 T    N     F     I     K1    K2    MD
---  ----  ---  -----  ----  ----  ----
600  2400   4   48600   4     2     6
=======================================

Formulations:
=================================================================
       Column Indices:             0        1        2        3
-----------------------------  ---------  -----  ---------  -----
 X1: Linear Characteristics     prices      x    satellite  wired
X2: Nonlinear Characteristics  satellite  wired
=================================================================
```

[31]:
```
# now we resolve the problem given the optimal instruments
```

```
demand_problem_results = demand_problem_w_instruments.solve(sigma=0.99*np.
    ↪identity(2),optimization=pyblp.Optimization('l-bfgs-b', {'maxls': 30}))
```

```
Solving the problem ...

Nonlinear Coefficient Initial Values:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +9.900000E-01
  wired    +0.000000E+00  +9.900000E-01
=======================================

Nonlinear Coefficient Lower Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +0.000000E+00
  wired    +0.000000E+00  +0.000000E+00
=======================================

Nonlinear Coefficient Upper Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite      +INF
  wired    +0.000000E+00      +INF
=======================================

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations    Evaluations  Iterations   Evaluations  Shares       Value
Improvement    Gradient Norm             Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  ----------------------------
 1          0             1          4441         13671        0
-4.871846E+16                 +6.644277E+09  +9.900000E-01, +9.900000E-01
 1          0             2          3111         9787         0
-8.257676E+16  +3.385830E+16  +4.869321E+09  +2.828932E-01, +2.828932E-01
```

```
 1         1             3            0            600         0
-4.918336E+16                +5.388201E-08  +0.000000E+00, +0.000000E+00
 1         1             4         3112           9794         0
-3.109528E+16                +5.839840E+09  +2.828932E-01, +2.828932E-01
 1         1             5         3111           9787         0
-8.257676E+16                +4.869321E+09  +2.828932E-01, +2.828932E-01
 1         1             6         3109           9791         0
-4.857029E+16                +3.903649E+09  +2.828932E-01, +2.828932E-01
 1         1             7         3111           9787         0
-8.257676E+16                +4.869321E+09  +2.828932E-01, +2.828932E-01

Optimization completed after 00:00:11.
Computing the Hessian and and updating the weighting matrix ...
Computed results after 00:00:08.

Problem Results Summary:
================================================================================
===============
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------
 1    -8.257676E+16  +4.869321E+09   -2.349511E+17    -2.874343E+16      0
+8.124302E+16
================================================================================
===============

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm             Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  ----------------------------
 2         0             1            0            600         0
+6.537189E-13                +1.091195E-12  +2.828932E-01, +2.828932E-01

Optimization completed after 00:00:01.
```

```
Computing the Hessian and estimating standard errors ...
Computed results after 00:00:05.

Problem Results Summary:
===============================================================================
==================================
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix  Covariance Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number  Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------  -----------------
 2    +6.537189E-13  +1.091195E-12   -2.784026E-12    +4.671904E-12      0
+2.505258E+17      +1.548412E+17
===============================================================================
==================================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:25       Yes          2            10          19995        63817
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
===========================================
 Sigma:       satellite          wired
---------  ---------------  ---------------
satellite   +2.828932E-01
           (+1.012906E+00)

  wired     +0.000000E+00    +2.828932E-01
                            (+1.039249E+00)
===========================================

Beta Estimates (Robust SEs in Parentheses):
=================================================================
    prices             x           satellite         wired
---------------  ---------------  ---------------  ---------------
 +1.217125E+00    +5.763916E-01    -4.337985E+00    -4.407183E+00
```

```
(+2.509104E-02)  (+5.076744E-02)  (+1.455546E-01)  (+1.485866E-01)
==================================================================
```

These estimates are not bad.

1.2 5.a: Demand and Supply Estimation

[32]:

```python
full_problem = pyblp.Problem(
    [
        pyblp.Formulation("0 + prices + x + satellite + wired"),
        pyblp.Formulation("0 + satellite + wired"),
        pyblp.Formulation("1 + w")
    ],
    product_data = observed_data,
    integration=pyblp.Integration('product', size=9),
    costs_type="log"
)
```

```
Initializing the problem ...
Initialized the problem after 00:00:00.

Dimensions:
===================================================
 T    N     F    I     K1    K2    K3    MD    MS
---  ----  ---  -----  ----  ----  ----  ----  ----
600  2400   4   48600   4     2     2     3     2
===================================================

Formulations:
=================================================================
       Column Indices:            0        1        2        3
-----------------------------  ---------  -----  ---------  -----
 X1: Linear Characteristics     prices      x    satellite  wired
X2: Nonlinear Characteristics  satellite  wired
X3: Log Cost Characteristics       1        w
=================================================================
```

[33]:

```python
# once again, we construct optimal instruments
full_problem_w_instruments = full_problem.solve(sigma=np.
 ↪identity(2),beta=[-1,None,None,None]).compute_optimal_instruments().to_problem()
```

```
Solving the problem ...
```

```
Nonlinear Coefficient Initial Values:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +1.000000E+00
  wired    +0.000000E+00  +1.000000E+00
=======================================

Beta Initial Values:
==========================================================
   prices            x          satellite        wired
-------------  -------------  -------------  -------------
-1.000000E+00       NAN            NAN            NAN
==========================================================

Nonlinear Coefficient Lower Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +0.000000E+00
  wired    +0.000000E+00  +0.000000E+00
=======================================

Beta Lower Bounds:
==========================================================
   prices            x          satellite        wired
-------------  -------------  -------------  -------------
    -INF           -INF           -INF           -INF
==========================================================

Nonlinear Coefficient Upper Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite      +INF
  wired    +0.000000E+00      +INF
=======================================

Beta Upper Bounds:
==========================================================
```

```
   prices           x          satellite        wired
-------------  -------------  -------------  -------------
    +INF          +INF          +INF          +INF
=========================================================

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm                    Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------
-------------------------------------------
 1         0            1          4478         13776        0
+1.420645E-26                +2.916005E-11  +1.000000E+00, +1.000000E+00,
-1.000000E+00

Optimization completed after 00:00:03.
Computing the Hessian and and updating the weighting matrix ...
Computed results after 00:00:24.

Problem Results Summary:
================================================================================
===============
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------
 1    +1.420645E-26  +2.916005E-11   -5.950660E-04    +7.196896E-04      0
+1.338489E+01
================================================================================
===============

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
```

```
Improvement   Gradient Norm                     Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------
-------------------------------------------
 2         0             1            0           600         0
+1.213500E-25                 +1.303812E-11  +1.000000E+00, +1.000000E+00,
-1.000000E+00

Optimization completed after 00:00:02.
Computing the Hessian and estimating standard errors ...
Computed results after 00:00:18.

Problem Results Summary:
================================================================================
==================================
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix  Covariance Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number  Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------  -----------------
 2    +1.213500E-25  +1.303812E-11   -6.052454E-03    +5.830682E-04      0
+7.176547E+01      +1.366273E+18
================================================================================
==================================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:47       Yes          0             4          4478         14376
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
===========================================
 Sigma:       satellite          wired
---------  ---------------  ---------------
satellite   +1.000000E+00
           (+4.250962E-03)
```

```
  wired     +0.000000E+00    +1.000000E+00
                            (+4.192555E-03)
===========================================

Beta Estimates (Robust SEs in Parentheses):
==================================================================
    prices             x            satellite          wired
---------------  ---------------  ---------------  ---------------
 -1.000000E+00    +9.090737E-01    +1.357618E+00    +1.341006E+00
(+8.317201E-03)  (+3.059498E-02)  (+2.117336E-02)  (+2.108796E-02)
==================================================================

Gamma Estimates (Robust SEs in Parentheses):
================================
       1                w
---------------  ---------------
 -1.406974E-01    +4.680814E-01
(+1.955098E-02)  (+1.085774E-02)
================================
Computing optimal instruments for theta ...
Computed optimal instruments after 00:00:04.

Optimal Instrument Results Summary:
==================================================
Computation  Error Term  Fixed Point  Contraction
   Time        Draws     Iterations   Evaluations
-----------  ----------  -----------  -----------
 00:00:04        1          9494         9494
==================================================
Re-creating the problem ...
Re-created the problem after 00:00:00.

Dimensions:
===================================================
 T    N     F     I     K1    K2    K3    MD    MS
---  ----  ---  -----  ----  ----  ----  ----  ----
600  2400   4   48600   4     2     2     7     8
===================================================

Formulations:
=================================================================
```

```
      Column Indices:            0        1        2        3
-----------------------------  ---------  -----  ---------  -----
 X1: Linear Characteristics     prices      x    satellite  wired
X2: Nonlinear Characteristics  satellite  wired
X3: Log Cost Characteristics       1        w
=================================================================
```

[34]:

```
# and here are the estimation results
full_problem_results = full_problem_w_instruments.solve(sigma=0.9*np.
 ↪identity(2),beta=[-1,None,None,None], check_optimality="both")
```

```
Solving the problem ...

Nonlinear Coefficient Initial Values:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +9.000000E-01
  wired    +0.000000E+00  +9.000000E-01
=======================================

Beta Initial Values:
==========================================================
   prices            x          satellite        wired
-------------  -------------  -------------  -------------
-1.000000E+00       NAN            NAN            NAN
==========================================================

Nonlinear Coefficient Lower Bounds:
=======================================
 Sigma:      satellite        wired
---------  -------------  -------------
satellite  +0.000000E+00
  wired    +0.000000E+00  +0.000000E+00
=======================================

Beta Lower Bounds:
==========================================================
   prices            x          satellite        wired
-------------  -------------  -------------  -------------
    -INF           -INF           -INF           -INF
==========================================================
```

```
Nonlinear Coefficient Upper Bounds:
=======================================
 Sigma:      satellite       wired
---------  -------------  -------------
satellite      +INF
  wired    +0.000000E+00      +INF
=======================================

Beta Upper Bounds:
==========================================================
   prices           x          satellite       wired
-------------  -------------  -------------  -------------
    +INF          +INF           +INF           +INF
==========================================================

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm                     Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------
-------------------------------------------
 1         0            1          4336         13321        0
-2.855162E+02                 +9.659102E+02  +9.000000E-01, +9.000000E-01,
-1.000000E+00
 1         0            2          4336         13295        0
-4.184583E+03  +3.899067E+03  +1.464040E+03  +8.888484E-01, +8.888484E-01,
-1.999876E+00
 1         0            3          4274         13119        0
+2.238252E+03                 +9.920075E+02  +8.442422E-01, +8.442422E-01,
-5.999378E+00
 1         0            4          4329         13295        0
+1.038995E+04                 +1.309991E+03  +8.834613E-01, +8.834613E-01,
-2.482895E+00
 1         0            5          4337         13292        0
-8.407922E+02                 +3.796392E+02  +8.888062E-01, +8.888062E-01,
-2.003663E+00
 1         0            6          4336         13295        0
```

```
-4.184583E+03                 +1.464040E+03  +8.888484E-01, +8.888484E-01,
-1.999876E+00
 1        1            7            0           600        0
+4.393448E+08                 +6.001273E+05  +0.000000E+00, +0.000000E+00,
-1.466040E+03
 1        1            8           4329        13284       0
+1.010082E+04                 +1.417132E+03  +8.867007E-01, +8.867007E-01,
-5.537365E+00
 1        1            9           4333        13295       0
-4.623683E+03  +4.390999E+02  +1.030407E+03  +8.887269E-01, +8.887269E-01,
-2.200115E+00
 1        2           10           4442        13663       0
-7.730336E+02                 +1.507782E+03  +8.786028E-01, +1.025720E+00,
-2.753598E+00
 1        2           11           4341        13317       0
+6.407188E+03                 +2.506296E+03  +8.885155E-01, +8.915872E-01,
-2.211671E+00
 1        2           12           4335        13305       0
-1.187810E+04  +7.254420E+03  +3.797859E+02  +8.887268E-01, +8.887274E-01,
-2.200117E+00
 1        3           13           4353        13380       0
-4.620186E+03                 +1.877459E+03  +8.881199E-01, +8.972040E-01,
-2.203337E+00
 1        3           14           4334        13303       0
-4.355088E+03                 +2.911085E+03  +8.887268E-01, +8.887278E-01,
-2.200117E+00
 1        3           15           4336        13303       0
-4.623684E+03                 +1.970713E+03  +8.887268E-01, +8.887274E-01,
-2.200117E+00
 1        3           16           4335        13305       0
-1.187810E+04                 +3.797859E+02  +8.887268E-01, +8.887274E-01,
-2.200117E+00

Optimization completed after 00:00:50.
Computing the Hessian and and updating the weighting matrix ...
Computed results after 00:00:22.

Problem Results Summary:
================================================================================
===============
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
```

```
Weighting Matrix
Step      Value       Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------
 1    -1.187810E+04  +3.797859E+02   -2.719718E+10    +6.320185E+10      0
+5.553615E+18
===============================================================================
===============

Starting optimization ...

GMM   Optimization   Objective   Fixed Point  Contraction  Clipped    Objective
Objective      Projected
Step   Iterations   Evaluations  Iterations   Evaluations  Shares       Value
Improvement   Gradient Norm                    Theta
----  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------
-------------------------------------------
 2         0            1            0           600         0
+1.431873E+01                +8.875149E+01  +8.887268E-01, +8.887274E-01,
-2.200117E+00
 2         0            2           4353        13332        0
+1.171137E+02                +3.017598E+02  +1.046625E+00, +1.056654E+00,
-1.227045E+00
 2         0            3           4043        12373        0
+3.504647E+00  +1.081408E+01  +5.494750E+00  +9.263405E-01, +9.287298E-01,
-1.968317E+00
 2         1            4           4073        12481        0
+3.367158E+00  +1.374895E-01  +5.270739E+00  +9.381365E-01, +9.431001E-01,
-1.972367E+00
 2         1            5           4196        12890        0
+2.884910E+00  +4.822481E-01  +4.273512E+00  +9.853204E-01, +1.000581E+00,
-1.988567E+00
 2         2            6           4645        14209        0
+2.171413E+00  +7.134969E-01  +1.836328E+00  +1.207614E+00, +1.230418E+00,
-2.037731E+00
 2         3            7           4546        13907        0
+2.115084E+00  +5.632834E-02  +4.114676E-01  +1.166972E+00, +1.170077E+00,
-2.030503E+00
 2         4            8           4599        14044        0
```

```
+2.113129E+00  +1.955256E-03  +3.568824E-02  +1.169502E+00, +1.179589E+00,
-2.031245E+00
 2         5             9          4605         14059        0
+2.113105E+00  +2.412179E-05  +7.455956E-03  +1.168982E+00, +1.180300E+00,
-2.031264E+00
 2         6            10          4601         14058        0
+2.113104E+00  +6.821551E-07  +1.822522E-04  +1.168790E+00, +1.180292E+00,
-2.031252E+00
 2         7            11          4605         14056        0
+2.113104E+00  +4.416867E-11  +6.161568E-06  +1.168790E+00, +1.180292E+00,
-2.031252E+00
 2         8            12          4602         14057        0
+2.113104E+00  +1.625367E-13  +1.954205E-08  +1.168790E+00, +1.180292E+00,
-2.031252E+00
 2         9            13          4602         14056        0
+2.113104E+00  +5.240253E-14  +2.852862E-09  +1.168790E+00, +1.180292E+00,
-2.031252E+00

Optimization completed after 00:00:41.
Computing the Hessian and estimating standard errors ...
Computed results after 00:00:23.

Problem Results Summary:
================================================================================
==================================
GMM     Objective      Projected    Reduced Hessian  Reduced Hessian  Clipped
Weighting Matrix  Covariance Matrix
Step      Value      Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares
Condition Number  Condition Number
----  -------------  -------------  ---------------  ---------------  -------
----------------  -----------------
 2    +2.113104E+00  +2.852862E-09   +2.453329E+01    +3.924354E+02      0
+2.591693E+17      +2.872436E+04
================================================================================
==================================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
```

```
 00:02:16       Yes           14            31          118556       364494
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
===========================================
 Sigma:       satellite          wired
---------  ---------------  ---------------
satellite   +1.168790E+00
           (+2.422035E-01)

  wired     +0.000000E+00    +1.180292E+00
                            (+2.322707E-01)
===========================================

Beta Estimates (Robust SEs in Parentheses):
==================================================================
    prices              x            satellite          wired
---------------  ---------------  ---------------  ---------------
 -2.031252E+00    +1.051940E+00    +4.030990E+00    +4.036894E+00
(+8.741266E-02)  (+4.717634E-02)  (+2.140120E-01)  (+2.153075E-01)
==================================================================

Gamma Estimates (Robust SEs in Parentheses):
================================
       1                w
---------------  ---------------
 +4.862138E-01    +2.634093E-01
(+1.981867E-02)  (+1.109183E-02)
================================
```

These estimates are even better than the previous section. We'll use these in the coming sections.

1.3 5.b Own-price Elasticities, Diversion Ratios

[35]:
```
estimated_price_elasticities = full_problem_results.compute_elasticities()
```

```
Computing elasticities with respect to prices ...
Finished after 00:00:01.
```

[36]:
```
estimated_diversion_ratios = full_problem_results.compute_diversion_ratios()
```

```
Computing diversion ratios with respect to prices ...
Finished after 00:00:01.
```

[37]:
```python
estimated_own_price_elasticities = estimated_price_elasticities.reshape(T,J,J).
 ↪mean(axis=0)
```

[38]:
```python
true_price_elasticities.mean(axis=2), estimated_own_price_elasticities
```

[38]:
```
(array([[-4.06535006,  1.38543391,  0.80172334,  0.7895892 ],
        [ 1.27934133, -4.16553436,  0.71112989,  0.71512854],
        [ 0.73928313,  0.74163481, -4.17726162,  1.3416553 ],
        [ 0.72070405,  0.7189693 ,  1.30923805, -4.18978309]]),
 array([[-4.05026563,  1.36210624,  0.70040876,  0.66790244],
        [ 1.50046032, -4.1558555 ,  0.70040876,  0.66790244],
        [ 0.7378061 ,  0.65818679, -4.16101852,  1.38817984],
        [ 0.7378061 ,  0.65818679,  1.4468293 , -4.17569613]]))
```

The estimates are pretty close to the true values

[39]:
```python
estimated_diversion_ratios.reshape((T,J,J)).mean(axis=0)
```

[39]:
```
array([[0.32909482, 0.32544548, 0.17501464, 0.17044505],
       [0.3455732 , 0.3188287 , 0.17046779, 0.16513031],
       [0.18282689, 0.16629782, 0.3246221 , 0.32625318],
       [0.18145558, 0.16389933, 0.33358963, 0.32105546]])
```

[40]:
```python
true_diversion_ratios.mean(axis=2)
```

[40]:
```
array([[0.33115087, 0.30335128, 0.18522023, 0.18027762],
       [0.32317153, 0.32122579, 0.18063565, 0.17496703],
       [0.19329289, 0.17575241, 0.32765373, 0.30330097],
       [0.19192008, 0.17341037, 0.31037504, 0.32429451]])
```

These look reasonably close as well.

2 Part 6

[41]:
```python
# merge firms 1 and 2
observed_data['merger_1_ids'] = observed_data['firm_ids'].replace(2, 1)

# merge firms 1 and 3
```

```python
observed_data['merger_2_ids'] = observed_data['firm_ids'].replace(3, 1)
```

[42]:
```python
marginal_costs = full_problem_results.compute_costs()

merger_1_prices = full_problem_results.compute_prices(
    firm_ids=observed_data['merger_1_ids'],
    costs=marginal_costs
)

merger_2_prices = full_problem_results.compute_prices(
    firm_ids=observed_data['merger_2_ids'],
    costs=marginal_costs
)
```

```
Computing marginal costs ...
Finished after 00:00:01.

Solving for equilibrium prices ...
Finished after 00:00:03.

Solving for equilibrium prices ...
Finished after 00:00:03.
```

[43]:
```python
np.mean(eq_prices, axis=1)
```

[43]:
```
array([2.73266213, 2.71653207, 2.76078363, 2.73913598])
```

[44]:
```python
# relative price changes, merging 1 and 2
np.mean(merger_1_prices.reshape((T,J)),axis=0)
```

[44]:
```
array([2.97945002, 2.99353235, 2.77124927, 2.74875349])
```

[45]:
```python
# relative price changes, merging 1 and 3
np.mean(merger_2_prices.reshape((T,J)),axis=0)
```

[45]:
```
array([2.84694606, 2.72847439, 2.8831668 , 2.75133819])
```

[46]:
```python
reduction_factors = np.concatenate([0.85*np.ones([T,2]),np.ones([T,2])],axis=1).
 ↪reshape((T*J,1))
reduced_costs = marginal_costs * reduction_factors
```

```
merger_1_prices_w_cost_reduction = full_problem_results.compute_prices(
    firm_ids=observed_data['merger_1_ids'],
    costs=reduced_costs
)
```

```
Solving for equilibrium prices ...
Finished after 00:00:03.
```

[47]:
```
# post-merger relative price changes, 1 and 2 with marginal cost reduction
np.mean(merger_1_prices_w_cost_reduction.reshape((T,J)),axis=0)
```

[47]:
```
array([2.78201464, 2.79398463, 2.76110894, 2.73900857])
```

[48]:
```
pre_merger_surpluses = full_problem_results.compute_consumer_surpluses()
post_merger_surpluses = full_problem_results.
 ↪compute_consumer_surpluses(prices=merger_1_prices_w_cost_reduction)
```

```
Computing consumer surpluses with the equation that assumes away nonlinear
income effects ...
Finished after 00:00:01.

Computing consumer surpluses with the equation that assumes away nonlinear
income effects ...
Finished after 00:00:01.
```

[49]:
```
# assuming measure of consumers in each market is 1, the net surpluses are just the␣
 ↪sums
# this is the net effect on consumer welfare
np.sum(post_merger_surpluses - pre_merger_surpluses)
```

[49]:
```
-6.5718246112984176
```

[50]:
```
post_merger_shares = full_problem_results.
 ↪compute_shares(merger_1_prices_w_cost_reduction)
pre_merger_profits = full_problem_results.compute_profits()
post_merger_profits = full_problem_results.
 ↪compute_profits(merger_1_prices_w_cost_reduction, post_merger_shares, reduced_costs)
```

```
Computing shares ...
Finished after 00:00:00.
```

```
Computing profits ...
Finished after 00:00:01.

Computing profits ...
Finished after 00:00:01.
```

[51]:
```python
# once again assuming measure 1 of consumers in each market
# net change in profits
np.sum(post_merger_profits - pre_merger_profits)
```

[51]: 69.19100664228262

[52]:
```python
# welfare change
np.sum(post_merger_surpluses - pre_merger_surpluses) + np.sum(post_merger_profits -␣
 ↪pre_merger_profits)
```

[52]: 62.6191820309842