Solution

In [7]:
```
print(δ.reshape(-1,1))
```

```
[[-2.6 ]
 [-5.35]
 [-2.34]
 [-2.58]
 [-3.06]
 [-2.61]
 [-2.27]
 [-2.63]
 [-2.52]
 [-1.85]
 [-4.54]
 [-2.46]
 [-2.39]
 [-2.68]]
```

Berry, Levinsohn, Pakes (BLP) Model

- Indices -> j: product, t: market, i:customer

1. Random Coefficients Logit:

- Indirect Utility:
  - $u_{ijt} = x'_{jt} b_i + a_i * p_{jt} + \xi_{jt} + \epsilon_{ijt}$
  - x_{jt}: attributes of product j in market t
  - p_{jt}: price of product j in market t
  - ξ_{jt}: unobserved quality of product j in market t
  - ϵ_{ijt}: random Gumbel error (for ith customer, jth product in t market)
- Utility Maximization:
  - Customer i, in market t; chooses one product j^* out of all
  - $j^*_{it} = max_j[u_{ijt}]$
- Conditional Choice Probabilities:
  - Market Share: $s_{jt} = P(j^*_{it} = j) = \frac{\exp(x'_{jt} b_i + a_i * p_{jt} + \xi_{jt})}{1 + \sum_j (x'_{jt} b_i + a_i * p_{jt} + \xi_{jt})}$
  - Demand Derivatives:
    - $\frac{\partial s_j}{\partial p_k}|_{j \neq k} = a_i s_j s_k$
    - $\frac{\partial s_j}{\partial p_j} = a_i s_j (1 - s_j)$

2. Parameters to Shares

- Structural parameters
  - $[b_i; a_i] = [\beta; \alpha] + A * D_i + B * v_i$
  - D_i: demographics
  - $\theta_1 = [\beta; \alpha]$: "common preferences" for all customers
  - $\theta_2 = [A, B]$: "group-specific preferences"

- Constant-utility and random-utility:
  - $u_{ijt} = \delta_{jt} + \mu_{ijt}$
  - $\delta_{jt}(x_{jt}, p_{jt}, \xi_{jt}; \theta_1) = x'_{jt}\beta + \alpha * p_{jt} + \xi_{jt}$: Constant-utility (fixed for all customers, depends on product and market only)
  - $\mu_{ijt}(x_{jt}, p_{jt}, D_i, v_i; \theta_2) = [p_{jt}, x_{jt}]'(A * D_i + B * v_i)$: random-utility (varies for each customer)

- Market shares:
  - $s_{jt} = \sigma_{jt}(\delta_t, x_t, p_t; \theta_2) = \int_v \int_D \frac{\exp(\delta_{jt} + \mu_{ijt})}{1 + \sum_j (\delta_{jt} + \mu_{ijt})} dF(v) dF(D)$

- $\frac{\partial s_j}{\partial p_k} = \int\int \frac{P(j^*=i|j,t}{3}$

3. Inverting Demand

- Logic
  - We want to find the mean-utility δ_{jt} implied for any θ_2. We first find s_{jt} and μ_{it} implied by θ_2 and then find δ_{jt}.
  - $s_{jt} = \sigma_{jt}(\delta_t, x_t, p_t; \theta_2) = \int_v \int_D \frac{\exp(\delta_{jt}+\mu_{ijt})}{1+\sum_j(\delta_{jt}+\mu_{ijt})} dF(v)dF(D)$
  - $\delta_{jt} = \sigma_{jt}^{-1}(s_t, x_t, p_t; \theta_2) = x'_{jt}\beta + \alpha * p_{jt} + \xi_{jt}$ (once we have this we can estimate β and α by 2SLS.
- Algorithm:
  - 1. Guess $\theta_2 = [A, B]$ and set $k = 0$
  - 2. set $k = 0$ and use $\delta_{jt}^k = \log(s_{jt}) - \log(s_{0t})$
  - 3. Compute for each customer i, the probability to choose product j in t: $\frac{\exp(\delta_{jt}^k+\mu_{ijt})}{1+\sum_j(\delta_{jt}^k+\mu_{ijt})}$
  - 4. Avg over all customers to get the market share for product j in t: $\sigma_{jt}(\delta_t, x_t, p_t; \theta_2) = (1/ns)\sum \frac{\exp(\delta_{jt}^k+\mu_{ijt})}{1+\sum_j(\delta_{jt}^k+\mu_{ijt})}$
  - 5. Apply contraction mapping: $exp(\delta_{jt}^{k+1}) = exp(\delta_{jt}^k)\frac{s_{jt}}{\sigma_{jt}(\delta_t^k, x_t, p_t; \theta_2)}$
  - 6. set $k = k + 1$ and go back to 3 until $\delta^{k+1} - \delta^k$ below tolerance.

In [1]:
```
import pandas as pd
import numpy as np
np.set_printoptions(precision=2)
df = pd.read_csv('/Users/pranjal/Desktop/Structural-Economics/io/random-coefficients-logit/
df
```

Out[1]:

	cdid	prodid	s_jt	cons1	cons2	cons3	cons4	cons5	cons6	cons7	...	cons41	con
0	1	1	0.046474	0.045385	-0.140034	0.079154	-0.161694	0.247079	-0.082650	-0.481462	...	-0.359403	-0.359
1	1	2	0.002790	0.449763	0.442506	-0.421718	0.435539	0.437777	-0.129650	-0.304529	...	-0.348173	-0.102
2	1	3	0.062422	0.462798	-0.382640	0.488984	0.235094	-0.223185	0.240559	-0.325140	...	0.201980	0.014
3	1	4	0.049676	0.269140	0.274197	-0.434930	0.286413	-0.261340	0.417779	0.498312	...	0.399959	-0.153
4	1	5	0.029658	0.441156	0.392380	0.080439	0.266950	0.200849	0.433510	-0.300214	...	-0.450249	0.108
5	1	6	0.047682	0.211303	0.426995	-0.242505	0.393396	-0.424232	-0.102105	0.244284	...	0.310010	0.353
6	1	7	0.066592	-0.315850	0.092849	0.024730	0.115274	0.221220	-0.328235	0.326760	...	0.491916	0.110
7	1	8	0.046740	0.271659	0.033336	-0.186894	0.254425	-0.297300	0.388584	0.276544	...	0.335080	0.047
8	1	9	0.047965	-0.100122	-0.066365	-0.365855	-0.089553	-0.159405	0.033011	0.345223	...	0.293160	-0.467
9	2	1	0.113105	0.323404	0.203230	0.297017	0.222407	0.463325	0.182649	-0.141104	...	0.173135	-0.492
10	2	3	0.007645	-0.099325	-0.361114	-0.351154	0.479626	-0.398829	-0.475233	-0.431773	...	0.230829	0.340
11	2	5	0.060103	-0.169256	-0.225614	-0.392618	-0.352281	0.241298	-0.231837	-0.213565	...	-0.097024	0.484
12	2	7	0.068611	0.232630	0.283477	-0.456474	0.294299	0.449266	0.246000	-0.012402	...	0.370328	0.286
13	2	9	0.050535	0.240073	0.476419	0.455113	-0.278779	-0.213454	0.344155	0.294992	...	0.398418	-0.460

14 rows × 53 columns

In [2]:
```
# index of records
ID = np.array(df.index)
ID_idx = ID.shape[0]

# cdid: market id (total 2)
cdid = np.array(df['cdid'])

# prodid: product id (total 9)
prodid = np.array(df['prodid'])

# cdindex: index of last element in market
cdindex = np.searchsorted(cdid, np.unique(cdid))

# market shares for each product j and market t
s = np.array(df['s_jt'])

# Mean-Deviations: μ_ijt for ith customer, for JxT product/markets.
μ = np.array(df.drop(['cdid', 'prodid', 's_jt'], axis = 1))

# Share of the outside good in each market t
s_sum = np.array(df[['s_jt','cdid']].groupby('cdid').sum())
s0 = 1 - s_sum
s0 = np.where(cdid==1, s0[0], s0[1])

# Initial guess for Mean-utilities
δ = np.log(s) - np.log(s0)
print(δ)

# Number of customers, products and markets
N = μ.shape[1]
J = np.unique(prodid).shape[0]
T = np.unique(cdid).shape[0]
```

```
[-2.56 -5.37 -2.26 -2.49 -3.01 -2.53 -2.2  -2.55 -2.53 -1.82 -4.52 -2.46
 -2.32 -2.63]
```

In [3]:
```
def CCP(μ_ijt, μ_irt, δ_jt, δ_rt):
    '''For a market t, given mean valuations and mean deviations of products '''
    try:
        return (np.exp(δ_jt + μ_ijt)/(1 + np.sum(np.exp(δ_rt + μ_irt))))[0]
    except:
        return 0

# Example
i = 1
j = 1
t = 2
idx = np.multiply(cdid==t, prodid==j)
print(idx)
μ_ijt = μ[idx, i-1] # scalar: mean deviations for i,j,t
μ_irt = μ[cdid==t, i-1] # vector: mean deviations for i, t for all products
δ_jt = δ[idx] # scalar: mean-valuation for j,t (fixed for all customers)
δ_rt = δ[cdid==t] # vector: mean-valuation for all products in market t
print(μ_ijt.shape, μ_irt.shape, δ_jt.shape, δ_rt.shape)
print(CCP(μ_ijt, μ_irt, δ_jt, δ_rt))
```

```
[False False False False False False False False False  True False False
 False False]
(1,) (5,) (1,) (5,)
0.14678166923652114
```

In [4]:
```
def CCPMatrix(μ, δ):
    '''Return Consumer Choice Probability for each i and product/market'''
    P = np.zeros((ID_idx, N))
    for t in range(1,T+1):
        for j in range(1,J+1):
            idx = np.multiply(cdid==t, prodid==j)
            δ_jt = δ[idx]
            δ_rt = δ[cdid==t]
            for i in range(1,N+1):
                μ_ijt = μ[idx, i-1]
                μ_irt = μ[cdid==t, i-1]
                P[idx, i-1] = CCP(μ_ijt, μ_irt, δ_jt, δ_rt)
    return P

P = CCPMatrix(μ, δ)
print(P.shape)

# Checks
idx = np.multiply(cdid==2, prodid==3)
print(idx)
print(np.sum(P[idx, :]/50), np.sum(s[ID[idx]]))

idx = np.multiply(cdid==2, prodid==4)
print(idx)
print(np.sum(P[idx, :]/50), np.sum(s[ID[idx]]))
```

```
(14, 50)
[False False False False False False False False False False  True False
 False False]
0.007735102054792866 0.007645334
[False False False False False False False False False False False False
 False False]
0.0 0.0
```

In [5]:
```
def σ_jt(P, j, t):
    '''Using CCP return Market share for product j and t'''
    idx = np.multiply(cdid==t, prodid==j)
    if np.mean(P[idx, :])>0:
        return np.mean(P[idx, :])
    else:
        return 0

# Checks
print(σ_jt(P, 1, 1))
print(σ_jt(P, 4, 2))
```

```
0.047173663262102906
0

/usr/local/lib/python3.10/site-packages/numpy/core/fromnumeric.py:3432: RuntimeWarning: Me
an of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/usr/local/lib/python3.10/site-packages/numpy/core/_methods.py:190: RuntimeWarning: invali
d value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

In [6]:
```
def contractionMap(δ, μ, tol=0.000001):
    '''Input: Guess for mean-valuations and mean-deviations for all products and all market
    Output: Optimal mean-valuations
    '''
    expδ = np.exp(δ)
    error = 1
    cnt = 1
    while error > tol:
        print(cnt)
        P = CCPMatrix(μ, np.log(expδ))
        for t in range(1,T+1):
            for j in range(1,J+1):
                idx = np.multiply(cdid==t, prodid==j)
                expδ[idx] = expδ[idx]*s[idx]/σ_jt(P, j, t)
                error = np.linalg.norm(expδ[idx]*s[idx]/σ_jt(P, j, t) - expδ[idx])
        cnt = cnt + 1
    return np.log(expδ) # return δ

δ_0 = np.log(s) - np.log(s0) # initial guess
δ = contractionMap(δ_0, μ)

print(δ)
```

```
1
2
3
4
5
6
7
8
[-2.6  -5.35 -2.34 -2.58 -3.06 -2.61 -2.27 -2.63 -2.52 -1.85 -4.54 -2.46
 -2.39 -2.68]
```

In [ ]:

In [ ]: