BLP PS Solution and Python Outputs

Hude Hude hh3024

II. Data

B. Summary statistics of the dataset [4 points]

i. [1 point] Read in the dataset

```
# i) Read Dataset
nevo_product_data = pd.read_csv(pyblp.data.NEVO_PRODUCTS_LOCATION)
```

ii. [1 point] By using the head() function, report the list of variables and the first five rows in the data, and verify that the dataset has the variables described in the previous section.

```
# ii) View Dataset
print(nevo_product_data.head())
```

```
  market_ids  city_ids  quarter product_ids  firm_ids  brand_ids    shares  \
0      C01Q1         1        1       F1B04         1          4  0.012417
1      C01Q1         1        1       F1B06         1          6  0.007809
2      C01Q1         1        1       F1B07         1          7  0.012995
3      C01Q1         1        1       F1B09         1          9  0.005770
4      C01Q1         1        1       F1B11         1         11  0.017934

     prices  sugar  mushy  ...  demand_instruments10  demand_instruments11  \
0  0.072088      2      1  ...              2.116358             -0.154708
1  0.114178     18      1  ...             -7.374091             -0.576412
2  0.132391      4      1  ...              2.187872             -0.207346
3  0.130344      3      0  ...              2.704576              0.040748
4  0.154823     12      0  ...              1.261242              0.034836

   demand_instruments12  demand_instruments13  demand_instruments14  \
0             -0.005796              0.014538              0.126244
1              0.012991              0.076143              0.029736
2              0.003509              0.091781              0.163773
3             -0.003724              0.094732              0.135274
4             -0.000568              0.102451              0.130640

   demand_instruments15  demand_instruments16  demand_instruments17  \
0              0.067345              0.068423              0.034800
1              0.087867              0.110501              0.087784
2              0.111881              0.108226              0.086439
3              0.088090              0.101767              0.101777
4              0.084818              0.101075              0.125169

   demand_instruments18  demand_instruments19
0              0.126346              0.035484
1              0.049872              0.072579
2              0.122347              0.101842
3              0.110741              0.104332
4              0.133464              0.121111

[5 rows x 30 columns]
```

iii. [1 point] Verify that the dataset has 2,256 observations and it contains information on: 47 cities; 2 quarters; 23 brands; 94 markets.

```
# iii) Verify Dataset
total_rows = nevo_product_data.shape[0]
num_cities = nevo_product_data['city_ids'].nunique()
num_quarters = nevo_product_data['quarter'].nunique()
num_brands = nevo_product_data['product_ids'].nunique()
num_markets = nevo_product_data['market_ids'].nunique()
```

```
Total number of observations: 2256
Unique cities: 47
Unique quarters: 2
Unique brands: 23
Unique markets: 94
```

iv. [1 point] Find the mean, median, and standard deviation of prices, shares, sugar, and mushy variables.

```
# 4) Summary of Statistics
print(nevo_product_data.describe())
```

```
          city_ids      quarter    firm_ids    brand_ids       shares  \
count  2256.000000  2256.000000  2256.000000  2256.000000  2256.000000
mean     32.340426     1.500000     2.125000    17.041667     0.019825
std      17.674998     0.500111     1.268893    12.610594     0.025600
min       1.000000     1.000000     1.000000     2.000000     0.000182
25%      16.000000     1.000000     1.000000     7.750000     0.005183
50%      33.000000     1.500000     2.000000    13.500000     0.011141
75%      47.000000     2.000000     2.250000    20.750000     0.024646
max      65.000000     2.000000     6.000000    48.000000     0.446883

            prices        sugar       mushy  demand_instruments0  \
count  2256.000000  2256.000000  2256.000000          2256.000000
mean      0.125740     8.625000     0.333333             0.025432
std       0.029035     5.787851     0.471509             0.375534
min       0.045487     0.000000     0.000000            -1.069056
25%       0.105491     3.000000     0.000000            -0.212389
50%       0.123829     8.500000     0.000000             0.030063
75%       0.143293    13.250000     1.000000             0.232839
max       0.225728    20.000000     1.000000             1.090208

       demand_instruments1  ...  demand_instruments10  demand_instruments11  \
count          2256.000000  ...           2256.000000           2256.000000
mean             -0.002419  ...             -0.515369             -0.144608
std               0.045155  ...              3.990604              0.236072
min              -0.195411  ...            -17.193444             -0.915056
25%              -0.023902  ...             -2.865767             -0.301611
50%               0.001283  ...             -0.271513             -0.038457
75%               0.026311  ...              2.086112              0.013594
max               0.122818  ...             10.193893              0.423279

       demand_instruments12  demand_instruments13  demand_instruments14  \
count           2256.000000           2256.000000           2256.000000
mean               0.003708              0.090780              0.091614
std                0.016006              0.030867              0.058537
min               -0.086395              0.003210             -0.164910
25%               -0.005808              0.070321              0.051978
50%                0.004256              0.088880              0.092130
75%                0.015140              0.109991              0.130278
max                0.049963              0.206038              0.283321
```

```
       demand_instruments15  demand_instruments16  demand_instruments17  \
count           2256.000000           2256.000000           2256.000000
mean               0.090725              0.091702              0.090843
std                0.051407              0.042725              0.030745
min               -0.070215             -0.049078             -0.003689
25%                0.056119              0.061993              0.070375
50%                0.090167              0.092052              0.088712
75%                0.125697              0.121167              0.109972
max                0.271270              0.253372              0.206615

       demand_instruments18  demand_instruments19
count           2256.000000           2256.000000
mean               0.091712              0.090812
std                0.054232              0.029365
min               -0.083744             -0.002656
25%                0.054265              0.070948
50%                0.091365              0.089010
75%                0.128666              0.109260
max                0.291082              0.188043

[8 rows x 28 columns]
```

III. Demand Estimation
A. Multinomial Logit [3 points]

i. Use the Formulation() and Problem() functions from the pyblp package to set up the multinomial logit problem.

a. [1 point] Use the Formulation() function to define which fields from the product_data you read into Spyder capture the price of products and the product identifiers based on which you define the product fixed effects.

```
# i.a) Formulation
logit_formulation = pyblp.Formulation('prices', absorb='C(product_ids)')
print(logit_formulation)

prices + Absorb[C(product_ids)]
```

b. [1 point] Use the Problem() function to define, based on the formulation you set up, the multinomial logit problem. Display the properties of the problem and in particular confirm that the data contains (i) 94 markets; (ii) 2,256 observations; (iii) 5 firms; (iv) number of linear demand parameters; (v) number of instrumental variables; (vi) number of variables based on which we define fixed effects; (viii) number of linear component of the model when abstracting from price.

```
# i.b) Problem
problem = pyblp.Problem(logit_formulation, nevo_product_data)


Initializing the problem ...
Absorbing demand-side fixed effects ...
Initialized the problem after 00:00:00.
```

```
Dimensions:
================================
 T    N     F    K1    MD    ED
---  ----  ---  ----  ----  ----
94   2256   5    1     20    1
================================

Formulations:
===================================
     Column Indices:          0
--------------------------  ------
X1: Linear Characteristics  prices
===================================
```

ii. [1 point] Use the problem.solve() function from the pyblp package to estimate the multinomial logit demand system based on the formulation and problem you set up in i. above.
a. Display the estimation results and specify: (i) estimation method used; (ii) value of the objective function; (iii) estimated α coefficient on price in the multinomial logit model we described above and the standard error of this; (iv) does α have the sign we would expect based on theory?

```
# ii.a) Solve
logit_results = problem.solve()

Solving the problem ...
Updating the weighting matrix ...
Computed results after 00:00:00.

Problem Results Summary:
==============================================
GMM     Objective    Clipped  Weighting Matrix
Step      Value      Shares   Condition Number
----  -------------  -------  ----------------
 1    +1.899432E+02     0      +6.927228E+07
==============================================

Estimating standard errors ...
Computed results after 00:00:00.

Problem Results Summary:
==============================================
GMM     Objective    Clipped  Weighting Matrix
Step      Value      Shares   Condition Number
----  -------------  -------  ----------------
 2    +1.874555E+02     0      +5.682065E+07
==============================================

Cumulative Statistics:
========================
Computation   Objective
   Time      Evaluations
-----------  -----------
 00:00:00         2
========================

Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -3.004710E+01
(+1.008589E+00)
===============
```

The final value of the objective function after the optimization process appears to be +1.874555E+02. The estimated α coefficient on price is -3.004710E+01.

The coefficient α associated with price reflects the sensitivity of the quantity demanded to changes in price. The demand for a product generally decreases as its price increases, implying that the price coefficient (α) should be negative.

In my results, the α coefficient is indeed negative (-30.04710), so it is consistent with the theory.

B. Nested Logit [5 points]

i. [1 point] Define a function called solve_nl that formulates and construct the problem for the nested logit estimation by using the Formulation(), Problem() and problem.solve() functions from the pyblp package, after defining the nesting_ids and the additional instruments by using the groupby(), groups() and transform() functions from the pandas package, and assuming that the initial value of ρ = 0.7.[4] The picture below shows the code for this function. Please explain what each line of this code does. The argument of the solve_nl function we just defined in the screenshot above is a dataset (i.e., df). Thus, as a next step, we will need to construct a dataset that we will pass through this function to estimate the nested logit system. We will do this in multiple steps.

```
# i)
def solve_nl(df):
    groups = df.groupby(['market_ids', 'nesting_ids'])
    df['demand_instruments20'] = groups['shares'].transform(np.size)
    nl_formulation = pyblp.Formulation('0 + prices')
    problem = pyblp.Problem(nl_formulation, df)
    return problem.solve(rho=0.7)
```

ii. [1 point] Define a new dataset df1 that is a copy of the product_data and add to this a nesting ID variable that puts all the products in the dataset in a single nest (i.e., add a new variable to df1 called nesting_ids which takes only value 1).

```
# ii)
df1 = nevo_product_data.copy()
df1['nesting_ids'] = 1
```

iii. [1 point] Apply solve_nl to df1 to estimate the nested logit model. Please summarize: (i) estimation method used; (ii) value of the objective function: (iii) estimated α coefficient on price and parameter ρ in the nested logit model we described above and the standard error of these; (iv) does α have the sign we would expect based on theory?; (v) how does the size of the estimated α coefficient compares to the estimates you obtained from the multinomial model estimation (i.e., III.A above); what could explain the differences you see?; (vi) when you inspect problem, how many instruments do you see?

```
# iii)
print("Solve a single nest case:")
nl_results1 = solve_nl(df1)
```

```
Solve a single nest case:
Initializing the problem ...
Initialized the problem after 00:00:00.
```

```
Dimensions:
===============================
 T    N     F    K1    MD    H
---  ----  ---  ----  ----  ---
94   2256   5    1     21    1
===============================

Formulations:
==================================
     Column Indices:          0
--------------------------  ------
X1: Linear Characteristics  prices
==================================
Solving the problem ...

Rho Initial Values:
=============
 All Groups
-------------
+7.000000E-01
=============

Rho Lower Bounds:
=============
 All Groups
-------------
+0.000000E+00
=============

Rho Upper Bounds:
=============
 All Groups
-------------
+9.900000E-01
=============

Starting optimization ...

GMM   Computation  Optimization   Objective   Fixed Point  Contraction  Clipped
Objective      Objective      Projected
Step     Time       Iterations   Evaluations  Iterations   Evaluations  Shares
Value       Improvement   Gradient Norm      Theta
----  -----------  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  -------------
 1     00:00:00         0            1            0            0           0
+1.331657E+02                 +6.086235E+02  +7.000000E-01
 1     00:00:00         0            2            0            0           0
+4.727024E+01  +8.589549E+01  +1.624078E+01  +9.900000E-01
 1     00:00:00         1            3            0            0           0
+4.720903E+01  +6.120631E-02  +1.528508E-09  +9.824626E-01

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

Problem Results Summary:
============================================================================
GMM     Objective      Projected       Reduced     Clipped  Weighting Matrix
Step      Value      Gradient Norm     Hessian     Shares   Condition Number
----  -------------  -------------  -------------  -------  ----------------
 1    +4.720903E+01  +1.528508E-09  +2.154684E+03     0      +1.995627E+09
============================================================================
```

```
Starting optimization ...

GMM   Computation  Optimization   Objective   Fixed Point  Contraction  Clipped
Objective      Objective      Projected
Step     Time       Iterations   Evaluations  Iterations   Evaluations  Shares
Value       Improvement   Gradient Norm      Theta
----  -----------  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  -------------
 2     00:00:00         0            1            0            0          0
+2.032711E+02                 +1.368046E+00  +9.824626E-01
 2     00:00:00         0            2            0            0          0
+2.035660E+02                 +7.961020E+01  +9.900000E-01
 2     00:00:00         0            3            0            0          0
+2.032711E+02  +8.710077E-05  +2.163302E-09  +9.825900E-01

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

Problem Results Summary:
=======================================================================================
=========
GMM     Objective      Projected       Reduced     Clipped  Weighting Matrix
Covariance Matrix
Step      Value      Gradient Norm     Hessian     Shares   Condition Number
Condition Number
----  -------------  -------------  -------------  -------  ----------------
-----------------
 2    +2.032711E+02  +2.163302E-09  +1.074355E+04     0      +2.004550E+09
+3.028074E+04
=======================================================================================
=========

Cumulative Statistics:
=================================================
Computation  Optimizer  Optimization   Objective
   Time      Converged   Iterations   Evaluations
-----------  ---------  ------------  -----------
 00:00:01       Yes          3             8
=================================================

Rho Estimates (Robust SEs in Parentheses):
===============
  All Groups
---------------
 +9.825900E-01
(+1.357591E-02)
===============

Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -1.173321E+00
(+3.971345E-01)
===============


print("Inspect problem:")
print(nl_results1.problem)


Inspect problem:
Dimensions:
```

```
================================
 T    N     F    K1    MD    H
---  ----  ---  ----  ----  ---
94   2256   5    1     21    1
================================

Formulations:
===================================
     Column Indices:          0
--------------------------  ------
X1: Linear Characteristics  prices
===================================
```

In theory, the price coefficient (α) in demand models is expected to be negative, as (α) represents the own-price elasticity. The estimated α is -1.173321E+00, which is negative. So the sign is consistent with the theory. In the multinomial logit model, the estimated *α* coefficient was − 30.04710, which is substantially larger in magnitude compared to −1.173321 from the nested logit model. The nested logit model accounts for similarities within groups of alternatives (nests), which can absorb some of the sensitivity to price that would otherwise be reflected in the *α* coefficient in a simple multinomial logit model. This can lead to a smaller magnitude of the price coefficient if the substitution patterns are partially captured by the nesting structure. Also, the nested logit model allows for correlation in unobserved factors within groups of choices (nests). This can make the model more flexible, thereby affecting the estimated coefficients.The model dimensions indicate *MD*=21. This suggests there are 21 instruments used in the estimation process.

iv. [1 point] Solve the same problem by defining two nests based on the mushy variable. One nest will contain all the observations with mushy = 1 and the other the rest of the observations with mushy = 0 in the product_data. For this, define a new dataset df2 that is a copy of product_data and add to this the new nesting ID variable defined, this time, based on mushy.

```
# iv)
df2 = nevo_product_data.copy()
df2['nesting_ids'] = df2['mushy']
```

v. [1 point] Apply solve_nl to df2 to estimate the nested logit model. Please summarize: (i) estimation method used; (ii) value of the objective function: (iii) estimated α coefficient on price and parameter ρ in the nested logit model we described above and the standard error of these; (iv) does α have the sign we would expect based on theory?; (v) how does the size of the estimated α coefficient compares to the estimates you obtained from the multinomial model estimation and that of the nested logit with only one nest (i.e., III.A and III.B.iii above)?; what could explain the differences you see?; (vi) when you inspect the problem you defined, how many instruments do you see?

```
# v)
nl_results2 = solve_nl(df2)

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

Dimensions:
================================
 T    N     F    K1    MD    H
---  ----  ---  ----  ----  ---
94   2256   5    1     21    2
================================

Formulations:
```

```
==================================
     Column Indices:          0
--------------------------  ------
X1: Linear Characteristics  prices
==================================
Solving the problem ...

Rho Initial Values:
=============
 All Groups
-------------
+7.000000E-01
=============

Rho Lower Bounds:
=============
 All Groups
-------------
+0.000000E+00
=============

Rho Upper Bounds:
=============
 All Groups
-------------
+9.900000E-01
=============
Starting optimization ...

GMM   Computation  Optimization   Objective   Fixed Point  Contraction  Clipped
Objective      Objective      Projected
Step     Time       Iterations   Evaluations  Iterations   Evaluations  Shares
Value       Improvement   Gradient Norm      Theta
----  -----------  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  -------------
 1     00:00:00         0            1            0            0           0
+3.552328E+02                  +4.602606E+02  +7.000000E-01
 1     00:00:00         0            2            0            0           0
+2.980378E+02  +5.719504E+01  +6.581208E+01  +9.900000E-01
 1     00:00:00         1            3            0            0           0
+2.968440E+02  +1.193805E+00  +2.196698E-09  +9.537208E-01

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

Problem Results Summary:
============================================================================
GMM     Objective      Projected       Reduced     Clipped  Weighting Matrix
Step      Value      Gradient Norm     Hessian     Shares   Condition Number
----  -------------  -------------  -------------  -------  ----------------
 1    +2.968440E+02  +2.196698E-09  +1.814068E+03     0      +6.747991E+08
============================================================================

Starting optimization ...

GMM   Computation  Optimization   Objective   Fixed Point  Contraction  Clipped
Objective      Objective      Projected
Step     Time       Iterations   Evaluations  Iterations   Evaluations  Shares
Value       Improvement   Gradient Norm      Theta
----  -----------  ------------  -----------  -----------  -----------  -------
-------------  -------------  -------------  -------------
 2     00:00:00         0            1            0            0           0
+7.011045E+02                  +3.488322E+02  +9.537208E-01
```

```
 2     00:00:00         0             2            0            0           0
+2.919897E+03                   +5.001750E+03  +0.000000E+00
 2     00:00:00         0             3            0            0           0
+6.902597E+02  +1.084484E+01  +6.396559E-09  +8.915428E-01

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

Problem Results Summary:
=======================================================================================
=========
GMM     Objective      Projected       Reduced     Clipped  Weighting Matrix
Covariance Matrix
Step       Value      Gradient Norm     Hessian     Shares   Condition Number
Condition Number
----  -------------  -------------  -------------  -------  ----------------
-----------------
 2    +6.902597E+02  +6.396559E-09  +5.610154E+03     0      +5.117503E+08
+1.992471E+04
=======================================================================================
=========

Cumulative Statistics:
=================================================
Computation  Optimizer  Optimization   Objective
   Time      Converged   Iterations   Evaluations
-----------  ---------  ------------  -----------
 00:00:01       Yes          3             8
=================================================

Rho Estimates (Robust SEs in Parentheses):
===============
  All Groups
---------------
 +8.915428E-01
(+1.913327E-02)
===============

Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -7.838283E+00
(+4.815462E-01)
===============
```

In the nested logit model estimation for df2, the estimated price coefficient (α) is -7.838283, which is consistent with economic theory; it indicates that an increase in price leads to a decrease in product demand, affirming the negative relationship predicted by the law of demand. This coefficient's magnitude lies between the more extreme value observed in the multinomial logit model -30.04710 and the milder effect seen in the nested logit model with one nest -1.173321. The differences in magnitude can be attributed to the models' varying abilities to account for substitution patterns and the complexity of their nesting structures, with the current model's two-nest structure providing a balance in capturing both product-specific sensitivities and broader category-level trends. The model uses 21 instruments, ensuring robust control for potential endogeneity between price and demand, enhancing the reliability of the estimated coefficients.

C. Random Coefficients Logit [13 points]

i. [1 point] Use the Formulation() command in pyblp to define the random coefficient model configuration both for the linear, X_1, and non-linear, X_2, characteristics determining demand.

a. In $X1$ include prices and fixed effects defined based on the product IDs so that your formulation allows for reporting estimates for each product fixed effect.

```
X1_formulation = pyblp.Formulation('0 + prices', absorb='C(product_ids)')
```

b. In X? include sugar, mushy, and prices (Note: product IDs are colinear with sugar or mushy, so we cannot include the product fixed effects as well in the vector of non-linear characteristics).

```
X2_formulation = pyblp.Formulation('1 + prices + sugar + mushy')
```

c. Formulate the random coefficient model based on $X1$ and $X2$ you constructed.

```
product_formulations = (X1_formulation, X2_formulation)
print(product_formulations)

(prices + Absorb[C(product_ids)], 1 + prices + sugar + mushy)
```

ii. [1 point] For defining the configuration of the integral over the distribution of the random coefficients, use the Integration() command from the pyblp package. This is constructed flexible enough that it accommodates multiple integration methods we reviewed in class. As a first approach for integration, use a Monte Carlo (i.e., denoted by mc) simulation method that constructs nodes and weights from a random normal distribution. Assume, as a starting point for the estimation, 50 individuals.

```
mc_integration = pyblp.Integration('monte_carlo', size=50,
specification_options={'seed': 0})
print(mc_integration)
Configured to construct nodes and weights with Monte Carlo simulation with options {seed:
0}.
```

iii. [1 point] Using the formulation and integration you defined in the previous two steps, and the product_data, define the random coefficient model based on the Problem() command from the pyblp package. Report the characteristics of this model such as:

a. Number of markets

b. Number of linear characteristics

c. Number of non-linear characteristics

```
mc_problem = pyblp.Problem(product_formulations, nevo_product_data,
integration=mc_integration)

Initializing the problem ...
Absorbing demand-side fixed effects ...
Initialized the problem after 00:00:00.

Dimensions:
============================================
 T    N     F    I     K1    K2    MD    ED
---  ----  ---  ----  ----  ----  ----  ----
94   2256   5   4700   1     4     20    1
============================================

Formulations:
```

```
==============================================================
       Column Indices:           0       1       2      3
-----------------------------  ------  ------  -----  -----
 X1: Linear Characteristics    prices
X2: Nonlinear Characteristics    1     prices  sugar  mushy
==============================================================
```

iv. [1 point] Configure the optimization algorithm you will use for the estimation of the random coefficient model by using the Optimization() command from the pyblp package. Use the Optimization() command to define the estimation algorithm and the corresponding tolerance level. As a first step, use a looser tolerance level so that the estimation routine runs faster (i.e., 1e-4) and the use the BFGS algorithm.

bfgs = pyblp.Optimization('bfgs', {'gtol': 1e-4})
print(bfgs)
Configured to optimize using the BFGS algorithm implemented in SciPy with analytic gradients and options {gtol: +1.000000E-04}.

v. [1 point] Apply the Solve() command in pyblp to the random coefficient problem you defined in step iii above by using the optimization configuration from step iv from above and by assuming that the initial value of the variance-covariance matrix is unrestricted, and takes values of only 1 (i.e., note that his can be defined by using np.ones command in Python).
Report the estimation results and specify:
a. Estimation method used and the value of objective function
b. Computation time of the estimation and after how many iterations convergence was achieved
c. Values of the estimated variance-covariance matrix of the multivariate normal distribution
d. Estimated coefficient and standard error of the coefficient of the linear characteristics

```
results1 = mc_problem.solve(sigma=np.ones((4, 4)), optimization=bfgs)
print(results1)

Problem Results Summary:
==============================================================================
==========================
GMM     Objective      Gradient         Hessian         Hessian     Clipped  Weighting
Matrix  Covariance Matrix
Step      Value          Norm       Min Eigenvalue  Max Eigenvalue  Shares   Condition
Number  Condition Number
----  -------------  -------------  --------------  --------------  -------
----------------  -----------------
 2    +1.483665E+02  +8.703749E-05  +8.522909E-02   +6.535574E+03      0
+5.150953E+07      +8.252073E+05
==============================================================================
==========================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:58       Yes          58           75          91080       279414
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
==============================================================================
===========================================================================
```

```
Sigma:          1              prices             sugar            mushy        |  Sigma
Squared:          1              prices             sugar            mushy
------  ---------------  ---------------  ---------------  ---------------  |
---------------  ---------------  ---------------  ---------------  ---------------
  1      +1.207566E+00                                                       |        1
+1.458216E+00    -1.382374E+01    +7.314899E-02    -7.099420E-01
        (+2.961805E+00)                                                      |
(+7.153152E+00)  (+5.188636E+01)  (+2.222540E-01)  (+2.272004E+00)
                                                                             |
prices   -1.144760E+01    +8.423600E+00                                      |
prices       -1.382374E+01    +2.020046E+02    -1.463091E+00    +1.492144E+00
        (+1.774956E+01)  (+1.157303E+01)                                     |
(+5.188636E+01)  (+3.052988E+02)  (+1.203963E+00)  (+1.508886E+01)
                                                                             |
sugar    +6.057555E-02    -9.136790E-02    +3.783374E-02                     |
sugar        +7.314899E-02    -1.463091E+00    +1.344888E-02    +2.034639E-02
        (+2.485504E-01)  (+2.282689E-01)  (+8.298206E-02)                    |
(+2.222540E-01)  (+1.203963E+00)  (+2.772954E-02)  (+2.731843E-01)
                                                                             |
mushy    -5.879114E-01    -6.218281E-01    -2.261704E-02    +4.800048E-01    |
mushy        -7.099420E-01    +1.492144E+00    +2.034639E-02    +9.632262E-01
        (+2.132815E+00)  (+1.532679E+00)  (+2.530561E+00)  (+1.330829E+00)  |
(+2.272004E+00)  (+1.508886E+01)  (+2.731843E-01)  (+3.964838E+00)
=====================================================================================
===========================================================================

Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -3.137350E+01
(+6.006485E+00)
===============
```

vi. [1 point] Repeat step v. from above by restricting this time the initial values of the variancecovariance matrix to the identity matrix (i.e., use np.eye command from Python to define the identity matrix)
Report the estimation results and specify:
a. Estimation method used and the value of objective function
b. Computation time of the estimation and after how many iterations convergence was achieved; How do these compare to the results you obtained in step b above where the initial value of the variance-covariance matrix was set to a matrix of all ones?
c. Values of the estimated variance-covariance matrix of the multivariate normal distribution; How do the values of this estimated variance-covariance matrix compare to the results you obtained in step b above where the initial value of the variance-covariance matrix was set to a matrix of all ones?
d. Estimated coefficient and standard error of the coefficient of the linear characteristics; Are the estimated coefficients smaller or larger than what you obtained in the previous step?

```
results3 = mc_problem.solve(sigma=np.eye(4), optimization=bfgs)
print(results3)

Problem Results Summary:
=====================================================================================
==========================
GMM     Objective      Gradient         Hessian          Hessian      Clipped  Weighting
Matrix  Covariance Matrix
Step      Value          Norm       Min Eigenvalue  Max Eigenvalue  Shares   Condition
Number  Condition Number
```

```
---- ------------- ------------- -------------- -------------- -------
---------------- -----------------
 2    +1.790116E+02  +1.252750E-06  +1.073316E+00   +6.045729E+03      0
+5.756956E+07      +4.755629E+04
=====================================================================================
==========================
```

```
Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:06       Yes          16           24          19535        60492
===========================================================================
```

```
Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
==========================================================================
Sigma:          1              prices            sugar             mushy
------  ---------------  ---------------  ---------------  ---------------
  1      +5.247994E-02
        (+1.144276E+00)

prices   +0.000000E+00    -4.338680E-01
                         (+8.013698E+00)

sugar    +0.000000E+00    +0.000000E+00    +3.563226E-02
                                          (+5.767988E-02)

mushy    +0.000000E+00    +0.000000E+00    +0.000000E+00    +5.010975E-01
                                                           (+1.706887E+00)
==========================================================================
```

```
Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -3.021224E+01
(+1.363484E+00)
===============
```

a. Estimation method used and the value of objective function

The estimation method used in both scenarios was the Generalized Method of Moments (GMM) within the pyblp framework, specifically employing the BFGS optimization algorithm. The estimation revealed significant differences in objective function values based on the initial settings of the variance-covariance matrix. When the matrix was initialized with all ones, the objective function value was approximately 148.3665. However, initializing the matrix with the identity matrix resulted in a higher objective function value of approximately 179.0116.

b. Computation time of the estimation and after how many iterations convergence was achieved; How do these compare to the results you obtained in step b above where the initial value of the variance-covariance matrix was set to a matrix of all ones?

The choice of initial values for the variance-covariance matrix had a notable impact on the efficiency of the estimation process. When the matrix was initialized with all ones, the computation took 58 seconds and required 58 iterations to converge. In contrast, initializing the matrix with the identity matrix significantly improved performance, reducing the computation time to just 6 seconds and the number of iterations to 16. This demonstrates that selecting appropriate initial values is crucial for optimizing the convergence speed and computational efficiency of model estimations.

c. Values of the estimated variance-covariance matrix of the multivariate normal distribution; How do the values of this estimated variance-covariance matrix compare to the results you obtained in step b above where the initial value of the variance-covariance matrix was set to a matrix of all ones?

In comparing the estimated values of the variance-covariance matrix from two different initial conditions in the pyblp framework, significant differences are observed. When initialized with all ones, the variance-covariance matrix showed considerable variability with diverse and higher magnitudes in both diagonal and off-diagonal elements, indicating complex interdependencies among variables. Conversely, initializing with the identity matrix resulted in more restrained and straightforward estimates, with many off-diagonal elements recorded as zero and lower values on the diagonal, suggesting minimal interaction between variables and more stable estimations.

d. Estimated coefficient and standard error of the coefficient of the linear characteristics; Are the estimated coefficients smaller or larger than what you obtained in the previous step?

With the matrix initialized as all ones, the estimated coefficient for prices was -31.37350 with a standard error of 6.006485. However, when initialized with the identity matrix, the coefficient was slightly smaller at -30.21224, but with a significantly reduced standard error of 1.363484. This reduction in the standard error suggests greater precision in the estimate when using the identity matrix as the initial condition.

vii. [1 point] Repeat steps ii-vi. from above by constructing nodes and weights based on a product rule that exactly integrates polynomials of degree 9 to approximate the integral instead of the Monte Carlo simulation method.
Report the estimation results (i.e., estimates of the variance and covariance matrix, and the coefficients and standard errors of the coefficients of the linear characteristics) and discuss how these compare to those obtained in the previous versions of the estimation you obtained based on the Monte Carlo simulation method you used to approximate the integral.

```
pr_integration = pyblp.Integration('product', size=5)
pr_problem = pyblp.Problem(product_formulations, nevo_product_data,
integration=pr_integration)
results2 = pr_problem.solve(sigma=np.ones((4, 4)), optimization=bfgs)
```

```
Problem Results Summary:
=================================================================================
=================
GMM   Objective  Gradient      Hessian         Hessian     Clipped  Weighting Matrix
Covariance Matrix
Step    Value      Norm    Min Eigenvalue  Max Eigenvalue  Shares   Condition Number
Condition Number
----  ---------  --------  --------------  --------------  -------  ----------------
-----------------
 2    +1.6E+02   +1.1E-05     +1.6E-02        +5.3E+03        0         +5.3E+07
+5.0E+20
=================================================================================
=================
```

```
Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:04:57       No           63           130         96884       301280
===========================================================================
```

```
Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
=====================================================================================================================
Sigma:      1         prices      sugar       mushy      |  Sigma Squared:      1
prices      sugar       mushy
------  ----------  ----------  ----------  ----------  |  --------------  ----------
----------  ----------  ----------
  1      -7.4E-01                                        |        1           +5.5E-01
-9.4E+00    +8.3E-02    -1.1E-01
        (+2.3E+00)                                       |                    (+3.4E+00)
(+3.5E+01)  (+1.6E-01)  (+6.4E-01)
                                                         |
prices   +1.3E+01    +6.6E-06                            |     prices         -9.4E+00
+1.6E+02    -1.4E+00    +1.9E+00
        (+7.5E+00)  (+2.7E+03)                           |                    (+3.5E+01)
(+1.9E+02)  (+8.0E-01)  (+8.9E+00)
                                                         |
sugar    -1.1E-01    -7.4E-08    -8.9E-09                |     sugar          +8.3E-02
-1.4E+00    +1.2E-02    -1.7E-02
        (+2.0E-01)  (+2.2E+05)  (+5.2E+04)               |                    (+1.6E-01)
(+8.0E-01)  (+2.2E-02)  (+1.6E-01)
                                                         |
mushy    +1.5E-01    -4.3E-07    +1.6E-07    +4.7E-08    |     mushy          -1.1E-01
+1.9E+00    -1.7E-02    +2.2E-02
        (+6.8E-01)  (+5.0E+03)  (+7.0E+02)  (+4.3E+02)   |                    (+6.4E-01)
(+8.9E+00)  (+1.6E-01)  (+2.0E-01)
=====================================================================================================================

Beta Estimates (Robust SEs in Parentheses):
==========
  prices
----------
 -3.1E+01
(+4.0E+00)
==========
```

In the pyblp model estimations using a product rule integration method, which exactly integrates polynomials of degree 9, the estimation results exhibit some distinct differences compared to those obtained from the Monte Carlo simulation method. The objective function value using product rule integration was slightly higher at 1.6 times 10^2, indicating a potentially less optimal fit compared to the Monte Carlo method. The variance-covariance matrix revealed a high level of precision with many of the variance and covariance estimates being quite close to zero, especially for off-diagonal entries, suggesting minimal interaction between some variables. The beta estimate for prices was -31 with a standard error of 4.0, showing consistent significant effects with a slightly reduced error margin compared to the Monte Carlo results.

This comparative analysis underscores how the choice of integration method impacts the precision and flexibility of model estimations. Product rule integration, with its exact nature, tends to offer more structured and accurate integration, potentially leading to more precise but possibly less flexible estimates of interactions among variables. In contrast, Monte Carlo simulations, with their inherent stochastic nature, might capture a broader range of dynamics due to random sampling, accommodating more variability and complexity in the model.

viii. [1 point] As a next step, we will add demographic characteristics to the estimation. For this, load into python the NEVO_AGENTS_LOCATION dataset and name this as agent_data.

By using the head() command in Python, report the first five rows of the agent_data, and confirm that the dataset contains the following variables:
a. market_ids that are the same as market_ids in the product_data we have been using so far.
b. city_ids
c. quarter
d. wights which are the weights attached to each individual consumer in the dataset. For each market, calculate the mean, standard deviation and sum of these weights? Do the weights sum to one for each market? If not, what do you observe regarding their distribution by market?
e. nodes0, nodes1, nodes2 are the nodes at which the unobserved individual tastes (i.e., denoted by μ?? in the theoretical model) are evaluated

f. demographic characteristics such income, income_squared, age and child. Please report the mean, standard deviation, median, 10th and 90th percentile of each of these demographic characteristics.

```
  market_ids  city_ids  quarter  ...  income_squared       age     child
0      C01Q1         1        1  ...        8.331304 -0.230109 -0.230851
1      C01Q1         1        1  ...        6.121865 -2.532694  0.769149
2      C01Q1         1        1  ...        1.030803 -0.006965 -0.230851
3      C01Q1         1        1  ...      -25.583605 -0.827946  0.769149
4      C01Q1         1        1  ...       -6.517009 -0.230109 -0.230851

[5 rows x 12 columns]

          city_ids      quarter  ...           age         child
count  1880.000000  1880.000000  ...  1.880000e+03  1.880000e+03
mean     32.340426     1.500000  ... -9.354220e-17  3.212560e-17
std      17.675782     0.500133  ...  9.445665e-01  4.214894e-01
min       1.000000     1.000000  ... -3.225841e+00 -2.308511e-01
25%      16.000000     1.000000  ... -3.926279e-01 -2.308511e-01
50%      33.000000     1.500000  ...  2.398946e-01 -2.308511e-01
75%      47.000000     2.000000  ...  6.453597e-01 -2.308511e-01
max      65.000000     2.000000  ...  1.273968e+00  7.691489e-01

[8 rows x 11 columns]

# Calculate the mean, standard deviation, and sum of the weights for each market
market_stats = agent_data.groupby('market_ids')['weights'].agg(['mean', 'std', 'sum'])

# Check if the weights sum to one for each market
market_stats['weights_sum_to_one'] = market_stats['sum'].apply(lambda x: abs(x - 1) <
1e-6)

print(market_stats)

            mean  std  sum  weights_sum_to_one
market_ids
C01Q1       0.05  0.0  1.0                True
C01Q2       0.05  0.0  1.0                True
C03Q1       0.05  0.0  1.0                True
C03Q2       0.05  0.0  1.0                True
C04Q1       0.05  0.0  1.0                True
...          ...  ...  ...                 ...
C61Q2       0.05  0.0  1.0                True
C63Q1       0.05  0.0  1.0                True
C63Q2       0.05  0.0  1.0                True
C65Q1       0.05  0.0  1.0                True
C65Q2       0.05  0.0  1.0                True

[94 rows x 4 columns]
```

```
# Calculate the sum of weights for each market
market_weight_sums = agent_data.groupby('market_ids')['weights'].sum()

# Check if the weights sum to one for each market
markets_not_summing_to_one = market_weight_sums[abs(market_weight_sums - 1) > 1e-6]

if not markets_not_summing_to_one.empty:
    print("Markets where weights do not sum to one:")
    print(markets_not_summing_to_one)
else:
    print("All market weights sum to one.")

All market weights sum to one.

# Calculate the descriptive statistics for each specified demographic characteristic
demographic_stats = agent_data[['income', 'income_squared', 'age',
print(demographic_stats)

              income  income_squared           age         child
count  1.880000e+03    1.880000e+03  1.880000e+03  1.880000e+03
mean  -7.440856e-16    7.105427e-15 -9.354220e-17  3.212560e-17
std    9.396363e-01    1.604345e+01  9.445665e-01  4.214894e-01
min   -4.938182e+00   -6.594640e+01 -3.225841e+00 -2.308511e-01
10%   -1.205887e+00   -2.127157e+01 -1.279931e+00 -2.308511e-01
50%    1.608203e-01    2.056507e+00  2.398946e-01 -2.308511e-01
90%    9.985438e-01    1.820214e+01  9.638135e-01  7.691489e-01
max    2.334590e+00    4.685632e+01  1.273968e+00  7.691489e-01
```

ix. [1 point] Use the Formulation() command from the pyblp package to define the agent_formulation which specifies which demographic characteristics you are interacting with the non-linear variables we included in X? in step i. Use income, income squared, age and child as such demographic characteristics.

```
agent_formulation = pyblp.Formulation('0 + income + income_squared + age + child')
print(agent_formulation)
income + income_squared + age + child
```

x. [1 point] Use the Problem() command from the pyblp package together with the agent_formulation, agent_data, product_formulations and product_data to set up the random coefficient model with observed individual characteristics to be estimated, consistent with Nevo (2000). Report the parameters of the problem you just defined and particularly specify what are the:
a. Linear characteristics
b. Non-linear characteristics
c. Demographics

```
nevo_problem = pyblp.Problem(
    product_formulations,
    nevo_product_data,
    agent_formulation,
    agent_data
)
print(nevo_problem)
Dimensions:
=================================================
 T    N     F    I     K1    K2    D    MD    ED
---  ----  ---  ----  ----  ----  ---  ----  ----
94   2256   5   1880   1     4     4    20    1
```

```
==================================================

Formulations:
===================================================================
       Column Indices:           0           1           2      3
-----------------------------  ------  --------------  -----  -----
 X1: Linear Characteristics    prices
X2: Nonlinear Characteristics    1         prices      sugar  mushy
       d: Demographics         income  income_squared   age   child
===================================================================
```

xi. [1 point] Define the initial values of the Σ element of the variance-covariance matrix (i.e., a diagonal matrix) and that of the Π matrix of the observed characteristics as in Nevo (2000).

```
initial_sigma = np.diag([0.3302, 2.4526, 0.0163, 0.2441])
initial_pi = np.array([
[ 5.4819, 0, 0.2037, 0 ],
[15.8935, -1.2000, 0, 2.6342],
[-0.2506, 0, 0.0511, 0 ],
[ 1.2650, 0, -0.8091, 0 ]
])
```

xii. [1 point] Set up the optimization algorithm based on the BFGS algorithm and with tolerance value of 1e-5 by using the Optimization() command from pyblp package.

```
tighter_bfgs = pyblp.Optimization('bfgs', {'gtol': 1e-5})

Configured to optimize using the BFGS algorithm implemented in SciPy with analytic
gradients and options {gtol: +1.000000E-05}.
```

xiii. [1 point] Use the Solve() command from the pyblp package, the initial values of the Σ and Π, and the optimization configuration you defined in the previous steps and a one-step GMM estimation method to estimate the random coefficient model with observed demographic characteristics.
Report the estimation results and specify:
a. Estimation method used and the value of objective function
b. Computation time of the estimation and after how many iterations convergence was achieved;
How do these compare to the results you obtained in the previously considered versions of the estimation?
c. Values of the estimated variance-covariance matrix of the multivariate normal distribution; How do these compare to the results you obtained in the previously considered versions of the estimation?
d. Estimated coefficient and standard error of the coefficient of the linear characteristics; How do these compare to the results you obtained in the previously considered versions of the estimation?

```
nevo_results = nevo_problem.solve(
    initial_sigma,
    initial_pi,
    optimization=tighter_bfgs,
    method='1s'
)
print(nevo_results)
Problem Results Summary:
```

```
=====================================================================================
==========================
GMM     Objective      Gradient         Hessian         Hessian     Clipped  Weighting
Matrix  Covariance Matrix
Step      Value          Norm       Min Eigenvalue  Max Eigenvalue  Shares   Condition
Number  Condition Number
----  -------------  -------------  --------------  --------------  -------
----------------  -----------------
 1    +4.561514E+00  +6.914885E-06  +2.380213E-05   +1.649684E+04      0
+6.927228E+07      +8.395896E+08
=====================================================================================
==========================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:00:39       Yes          51           57          46381       143962
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs in Parentheses):
=====================================================================================
=================================================================
Sigma:         1             prices           sugar           mushy       |   Pi:
income        income_squared          age             child
------  ---------------  ---------------  ---------------  ---------------  |  ------
---------------  ---------------  ---------------  ---------------
  1      +5.580936E-01                                                    |    1
+2.291971E+00    +0.000000E+00    +1.284432E+00    +0.000000E+00
        (+1.625326E-01)                                                   |
(+1.208569E+00)                   (+6.312149E-01)
                                                                          |
prices   +0.000000E+00    +3.312489E+00                                   |  prices
+5.883251E+02    -3.019201E+01    +0.000000E+00    +1.105463E+01
                         (+1.340183E+00)                                  |
(+2.704410E+02)  (+1.410123E+01)                   (+4.122564E+00)
                                                                          |
sugar    +0.000000E+00    +0.000000E+00    -5.783552E-03                  |  sugar
-3.849541E-01    +0.000000E+00    +5.223427E-02    +0.000000E+00
                                          (+1.350452E-02)                 |
(+1.214584E-01)                   (+2.598529E-02)
                                                                          |
mushy    +0.000000E+00    +0.000000E+00    +0.000000E+00    +9.341447E-02  |  mushy
+7.483723E-01    +0.000000E+00    -1.353393E+00    +0.000000E+00
                                                           (+1.854333E-01)  |
(+8.021081E-01)                   (+6.671086E-01)
=====================================================================================
=================================================================

Beta Estimates (Robust SEs in Parentheses):
===============
    prices
---------------
 -6.272990E+01
(+1.480321E+01)
===============
```

a. Estimation method used and the value of objective function

In the estimation using the pyblp package, a one-step Generalized Method of Moments (GMM) approach was employed to estimate the random coefficient model with observed

demographic characteristics. The objective function value achieved through this estimation method was reported as 4.561514.

b. Computation time of the estimation and after how many iterations convergence was achieved;How do these compare to the results you obtained in the previously considered versions of the estimation?

In the one-step GMM estimation method using the pyblp package, the computation took 39 seconds with convergence achieved in 51 iterations, representing a moderate performance when compared to previous estimation methods. Specifically, it was faster and required fewer iterations than the product rule integration, which took nearly 5 minutes and 63 iterations, and the Monte Carlo method with all ones, which took 58 seconds and 58 iterations. However, it was less efficient than the Monte Carlo method with the identity matrix, which concluded in just 6 seconds and 16 iterations. This indicates that the one-step GMM, while integrating demographic characteristics, offers a balanced approach, providing a comprehensive analysis without excessive computational demand, unlike the simpler or more complex previous methods.

c. Values of the estimated variance-covariance matrix of the multivariate normal distribution; How do these compare to the results you obtained in the previously considered versions of the estimation?

In the one-step GMM estimation using the pyblp package, the values of the estimated variance-covariance matrix of the multivariate normal distribution show significant diagonal entries +0.5580936 for the intercept term) with most off-diagonal terms estimated as zero, suggesting minimal correlation between the different characteristics within the model.

The variance-covariance matrix of the multivariate normal distribution revealed significant diagonal entries with most off-diagonal terms estimated as zero, indicating minimal inter-variable correlations. This result contrasts with earlier estimations where the Monte Carlo with all ones displayed more complex interdependencies with non-zero off-diagonal terms and larger diagonal values, and the Monte Carlo with identity matrix showed a simpler interaction pattern with smaller and zero off-diagonal values. Similarly, the product rule integration had moderate complexity in its variance-covariance matrix compared to the one-step GMM, which seems to streamline the estimation by focusing on direct effects with minimal interactions, thus providing a balanced approach in capturing the model's dynamics.

d. Estimated coefficient and standard error of the coefficient of the linear characteristics; How do these compare to the results you obtained in the previously considered versions of the estimation?

In the one-step GMM estimation using the pyblp package, the estimated coefficient for prices was significantly larger at -62.72990 with a robust standard error of 14.80321, compared to earlier methods. Specifically, this estimation shows a much larger effect of prices on the dependent variable than the Monte Carlo with all ones (-31.37350, SE 6.006485), Monte Carlo with identity matrix (-30.21224, SE 1.363484), and product rule integration (-31.00, SE 4.00). The higher standard error in the one-step GMM suggests increased variability in the estimate, which might be due to the complexity added by integrating demographic characteristics into the model. This indicates that while the one-step GMM method may capture a more pronounced impact of price changes, it also introduces greater uncertainty in the estimation process.