In [17]:
```
######################################
#Econs 514 -- Assignment 2 - BLP Rep Table 4
#Updated -- 2/21/23
#Due - 3/1/23
#By -- Suhina Deol
######################################
```

In [1]:
```
import os
import pandas as pd
import numpy as np

from os.path import dirname, join as pjoin
import scipy.io

#import scipy.optimize as opt

import statsmodels.api as sm
from statsmodels.iolib.summary2 import summary_col

from scipy.optimize import minimize
from itertools import combinations

from statsmodels.sandbox.regression.gmm import IV2SLS
```

In [3]:
```
######################################
#load data1
######################################
data = scipy.io.loadmat(r'iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/BLP
print(data.keys())

df=data['model_name']
#print(df)
#print(type(df))

######################################
#Change mat to csv
######################################
df=pd.DataFrame(df)
df=df.rename(columns={0: "model_name"})
df.to_csv('iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/df.csv')
```

```
dict_keys(['__header__', '__version__', '__globals__', 'model_name'])
```

In [4]:
```
######################################
#load data2
######################################
data2 = scipy.io.loadmat('iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/BLP
print(data2.keys())
df2=data2['model_id']

xtra_cols = ['share','outshr', 'const', 'mpd', 'mpg','air', 'space', 'hpwt', 'price',

for item in xtra_cols:
    df2 = np.concatenate([df2, data2[item]], axis=1)
df2 = pd.DataFrame(df2, columns = ['model_id'] + xtra_cols)

#print(df2)
```

```
######################################
#Change mat to csv
######################################
df2=pd.DataFrame(df2)
df2.to_csv('iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/df2.csv')
```

```
dict_keys(['__header__', '__version__', '__globals__', 'model_id', 'own_dummies', 'i
d', 'product', 'const', 'mpd', 'air', 'mpg', 'trend', 'space', 'hpwt', 'cdindex', 'cd
id', 'outshr', 'firmid', 'share', 'price'])
```

In [5]:
```
######################################
#Combine dataset
######################################
df3 = pd.concat([df,df2],axis = 1)
#print(df3)

#csv
df3=pd.DataFrame(df3)
df3.to_csv('iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/df3.csv')
```

In [6]:
```
######################################
#TABLE III
######################################
#COL1 - OLS
######################################
attributes = ['const', 'hpwt','air','mpd', 'space','price']
#print(attributes)

y = np.log(df3['share']/df3['outshr'])
#print(y)

xmat = df3[attributes].to_numpy()
#print(xmat)

#fit linear regression model
model = sm.OLS(y, xmat).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.387
Model:                            OLS   Adj. R-squared:                  0.386
Method:                 Least Squares   F-statistic:                     279.2
Date:                Fri, 17 Mar 2023   Prob (F-statistic):         6.52e-232
Time:                        18:35:13   Log-Likelihood:                -3319.4
No. Observations:                2217   AIC:                             6651.
Df Residuals:                    2211   BIC:                             6685.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -10.0716      0.253    -39.822      0.000     -10.568      -9.576
x1            -0.1243      0.277     -0.448      0.654      -0.668       0.419
x2            -0.0343      0.073     -0.472      0.637      -0.177       0.108
x3             0.2650      0.043      6.146      0.000       0.180       0.350
x4             2.3421      0.125     18.707      0.000       2.097       2.588
x5            -0.0886      0.004    -22.014      0.000      -0.097      -0.081
==============================================================================
Omnibus:                      158.000   Durbin-Watson:                   1.478
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              205.595
Skew:                          -0.632   Prob(JB):                     2.27e-45
Kurtosis:                       3.792   Cond. No.                         203.
==============================================================================

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

In [23]:

```
#####################################
#TABLE III
#####################################
#COL2 - Instruments for Price
#####################################

#stage1
attributes2 = ['mpd', 'air', 'space', 'hpwt']
#print(attributes)

y2 = df3['price']
#print(y)

zmat = df3[attributes2].to_numpy()
#print(xmat)

#fit linear regression model
results_fs = sm.OLS(y2, zmat).fit()

#view model summary
#print(results_fs.summary())

#stage2
df3['predicted_price'] = results_fs.predict()

attributes22 = ['const','hpwt','air','mpd', 'space','predicted_price']
#print(y)

xmat22 = df3[attributes22].to_numpy()
```

```python
results_ss = sm.OLS(y, xmat22).fit()

print(results_ss.summary())

#automatically with package with correct standard errors
iv = IV2SLS(y, xmat, zmat).fit()

print(iv.summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.253
Model:                            OLS   Adj. R-squared:                  0.251
Method:                 Least Squares   F-statistic:                     187.0
Date:                Fri, 03 Mar 2023   Prob (F-statistic):          3.37e-138
Time:                        14:25:28   Log-Likelihood:                -3539.1
No. Observations:                2217   AIC:                             7088.
Df Residuals:                    2212   BIC:                             7117.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -10.3658      0.279    -37.179      0.000     -10.913      -9.819
x1            -0.0013      0.044     -0.029      0.977      -0.088       0.086
x2            -0.0433      0.120     -0.362      0.718      -0.278       0.192
x3             0.3030      0.049      6.196      0.000       0.207       0.399
x4             2.4694      0.138     17.883      0.000       2.199       2.740
x5            -0.0887      0.008    -11.155      0.000      -0.104      -0.073
==============================================================================
Omnibus:                      118.524   Durbin-Watson:                   1.364
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              138.270
Skew:                          -0.573   Prob(JB):                     9.44e-31
Kurtosis:                       3.431   Cond. No.                     9.99e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The smallest eigenvalue is 4.13e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
<bound method IVRegressionResults.summary of <statsmodels.sandbox.regression.gmm.IVRe
gressionResults object at 0x000001B844B9D160>>
```

In [24]:

```python
#####################################
#TABLE III
#####################################
#COL3- Depvar ln(price)
#####################################
#Create log of hpwt and mpg
# apply log(x+1) element-wise to a subset of columns
log_hpwt = np.log(df3['hpwt'])
df3['log_hpwt'] = log_hpwt + 1

log_mpg = np.log(df3['mpg'])
df3['log_mpg'] = log_mpg + 1

log_space = np.log(df3['space'])
df3['log_space'] = log_space + 1
```

```python
attributes3 = ['const', 'log_hpwt','air','log_mpg', 'log_space','trend']
#print(attributes)

y3 = np.log(df3['price'])
#print(y)

xmat3 = df3[attributes3].to_numpy()
#print(xmat)

#fit linear regression model
model3 = sm.OLS(y3, xmat3).fit()

#view model summary
print(model3.summary())

#save updated dataset with log-vars
df3.to_csv('iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/df4.csv')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.656
Model:                            OLS   Adj. R-squared:                  0.656
Method:                 Least Squares   F-statistic:                     844.9
Date:                Fri, 03 Mar 2023   Prob (F-statistic):               0.00
Time:                        14:25:31   Log-Likelihood:                -567.01
No. Observations:                2217   AIC:                             1146.
Df Residuals:                    2211   BIC:                             1180.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.6184      0.149     17.563      0.000       2.326       2.911
x1             0.5203      0.035     14.833      0.000       0.452       0.589
x2             0.6798      0.019     36.247      0.000       0.643       0.717
x3            -0.4706      0.049     -9.694      0.000      -0.566      -0.375
x4             0.1248      0.063      1.967      0.049       0.000       0.249
x5             0.0128      0.002      8.526      0.000       0.010       0.016
==============================================================================
Omnibus:                      533.211   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1412.614
Skew:                           1.270   Prob(JB):                    1.80e-307
Kurtosis:                       5.974   Cond. No.                         307.
==============================================================================

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

In [75]:

```python
#####################################
#TABLE IV
#####################################
#BLP Specification
#Esimate parameters of demand and pricing equations
#####################################

class BLP(object):
    '''
    Jia Yan
```

```
    02/21/2023
    '''

    def __init__(self, path_data, file_data, path_str_data, file_str_data, ndraws=500,
        '''
        set parameters
        '''
        self.ndraws = ndraws # number of random draws in monte-carlo integration
        self.tol_fp = tol_fp # convergence tolerance level in nested fixed-point inter

        '''
        construct data
        '''
        self.df = self._initialize_data(path_data, file_data, path_str_data, file_str_
        self.nmarkets = int(self.df['trend'].max()) + 1
        self.y_fixed = np.log(self.df['share']/self.df['outshr']) # Dependent variable

        '''
        Demand model specification
        '''
        attributes = ['const', 'hpwt', 'air', 'mpd', 'space', 'price'] # demand side v
        self.attributes_random = ['const', 'hpwt', 'air', 'mpd', 'space'] # variables
        self.Xmat = self.df[attributes].to_numpy() # X-matrix in demand model

        '''
        construct demand-side instruments
        '''
        self.Zmat_D =  self._initialize_IV_D(self.df) # matrix of demand side instrume
        self.weight_mat_D = np.linalg.inv(np.matmul(np.transpose(self.Zmat_D), self.Zm
        self.project_mat = self._initialize_2sls_D() # inv(X'PX)(X'P) in which P = Z*i

        '''
        supply side specification
        '''
        mc = ['const', 'mpg', 'air', 'space', 'hpwt', 'trend'] # marginal cost variabl
        self.Mcmat = self._initialize_mc(mc) # matrix of variables in marginal cost eq

        '''
        construct supply-side instruments
        '''
        self.Zmat_s = self._initialize_IV_S()

        ''''
        Take standard normal draws for approximating integrals in market share and mar
        Better to use halton draws
        '''
        self.draws = np.random.randn(self.nmarkets, self.ndraws, len(self.attributes_r

    def _initialize_data(self, path_data, file_data, path_str_data, file_str_data):
        data = scipy.io.loadmat(os.path.join(path_data, file_data))
        model_name = scipy.io.loadmat(os.path.join(path_str_data, file_str_data))['mod
        v_list = ['outshr', 'const', 'mpd', 'mpg', 'air', 'space', 'hpwt', 'price', 't
        df = data['share']
        for item in v_list:
            df = np.concatenate([df, data[item]], axis=1)
        df = pd.DataFrame(df, columns = ['share'] + v_list)
        df['model_name'] = model_name
        df['maker'] = df['model_name'].transform(lambda x: x[0:2])
        df = df.sort_values(by=['trend', 'maker'], ascending=[True, True]) # group pro
        return df
```

```
    def _initialize_IV_D(self, df):
        # demand side instruments for price: sum of attributes of other products from
        z_list = ['const'] # creat instruments of price from this list
        IV_list = ['const', 'mpd', 'air', 'space', 'hpwt'] # the first part of IV are
        for var in z_list:
            name_own = var + "_" + "z" + "_" + "own"
            IV_list.append(name_own)
            name_rival = var + "_" + "z" + "_" + "rival"
            IV_list.append(name_rival)

            df[name_own] = df.groupby(['trend', 'maker'])[var].transform(lambda x: x.s
            df[name_own] = df[name_own] - self.df[var]

            df['junk']= df.groupby(['trend'])[var].transform(lambda x: x.sum()) - self
            df[name_rival] = df['junk'] - df[name_own]

        return df[IV_list].to_numpy() ## the matrix of demand-side instruments

    def _initialize_2sls_D(self):
        pz = np.matmul(self.Zmat_D, self.weight_mat_D)
        pz = np.matmul(pz, np.transpose(self.Zmat_D))
        project_mat = np.matmul(np.transpose(self.Xmat), pz)
        project_mat = np.matmul(project_mat, self.Xmat)
        project_mat = np.linalg.inv(project_mat)
        project_mat = np.matmul(project_mat, np.transpose(self.Xmat))
        return np.matmul(project_mat, pz)

    def _initialize_mc(self, vlist):
        df = self.df[vlist].copy()
        df['mpg'] = np.log(df['mpg']) +1
        df['space'] = np.log(df['space'])+1
        df['hpwt'] = np.log(df['hpwt'])+1
        return df.to_numpy()

    def _initialize_ols_S(self):
        mom = np.matmul(np.transpose(self.Mcmat), self.Mcmat)
        return np.matmul(np.linalg.inv(mom), self.Mcmat)

    def _initialize_IV_S(self):
        def util(pair):
            return np.multiply(pair[0], pair[1])
        m = self.Mcmat[:, 1:] # remove constant term
        l = [m[:,i] for i in range(m.shape[1])]
        pairs = combinations(l, 2)
        zmat = self.Mcmat # the first part of IVs are exo. regressors
        interactions = np.array(list(map(util, pairs)))
        interactions = np.transpose(interactions)
        return np.hstack((zmat, interactions))

    def ols(self):
        '''
        replicate the first column of table 3
        '''
        y = np.log(self.df['share']/self.df['outshr'])
        #b = np.matmul(np.transpose(self.Xmat), self.Xmat)
        #b = np.linalg.inv(b)
        #b = np.matmul(b, np.transpose(self.Xmat))
        #return np.matmul(b, self.y_fixed)
        return sm.OLS(self.y_fixed, self.Xmat).fit()
```

```
    def iv(self):
        '''
        replicate the second column of table 3
        '''
        #return np.matmul(self.project_mat, self.y_fixed)
        return IV2SLS(self.y_fixed, self.Xmat, self.Zmat_D).fit()

    def hedonic_price(self):
        '''replicate the third column of table 3'''
        y = np.log(self.df['price'])
        return sm.OLS(y, self.Mcmat).fit()

    def markup_conditional(self, udic):
        v = np.exp(udic['delta'] + (udic['rdraw'] * udic['xv']).sum(axis=1))
        s = v / (1 + np.sum(v))
        cut = 0
        dmat = []
        for i in udic['jf']:
            sg = s[cut:cut+i]
            dmat.append(np.diag(sg) - np.outer(sg,sg))
            cut = cut + i
        return {'dmat': dmat, 'shares': s}

    def share_conditional(self, udic):
        v = np.exp(udic['delta'] + (udic['rdraw'] * udic['xv']).sum(axis=1))
        return v / (1 + np.sum(v))

    def market_share(self, mid, delta, xv, jf, markup=False):
        draws = self.draws[mid]
        inputs = [{'delta': delta, 'xv':xv, 'rdraw':draws[r], 'jf': jf} for r in range
        if markup is False:
            out = map(self.share_conditional, inputs)
            return (1/self.ndraws) * np.sum(list(out), axis=0)
        else:
            out = map(self.markup_conditional, inputs)
            d = [i['dmat'] for i in out]
            s = [i['shares'] for i in out]
            dmat = [(1/self.ndraws) * item for item in map(sum, zip(*d))]
            shares = (1/self.ndraws) * np.sum(s, axis=0)
            return {'dmat': dmat, 'shares': shares}

    def fixed_point(self, pack):
        mid = pack['mid']
        df = pack['df']
        sigmas = pack['sigmas']
        s0 = df['share'].to_numpy()
        xv = sigmas * df[self.attributes_random].to_numpy()
        jf = [len(g) for _, g in df.groupby(['maker'])]
        check = 1.0
        delta_ini = np.zeros(len(s0))
        while check > self.tol_fp:
            delta_new = delta_ini + (np.log(s0) - np.log(self.market_share(mid, delta_
            check = np.max(abs(delta_new - delta_ini))
            delta_ini = delta_new

        '''
        Calculate markups in a market at current parameter values (delta, sigmas)
        '''
        mrkup = self.market_share(mid, delta_new, xv, jf, markup=True)
```

```
        return {'delta': delta_new, "markup": mrkup}

    def mean_utility(self,sigmas):
        """
        sigmas: an 1_D array with the shape (len(self.attributes_random), ), which cor
        the standard errors of random coefficients
        """
        df = self.df.copy()
        v_list = ['share', 'maker'] + self.attributes_random

        '''
        # step 1: solve mean utility (delta_j) from the fixed-point iteration
        '''
        df_list = [{'mid': int(mid), 'df': d[v_list], 'sigmas': sigmas} for mid, d in
        out = map(self.fixed_point, df_list)
        delta_j = tuple([i['delta'] for i in out])
        delta_j = np.concatenate(delta_j, axis=0) # an array with the shape(2217,)

        '''
        step 2: uncover mean part of coefficients (beta_bar) from delta_j, which is ec
        running an IV estimation using delta_j as the dependent variable
        '''
        beta_bar = np.matmul(self.project_mat, delta_j)

        '''
        step 3: uncover ommited product attributes (xi_j) from delta_j and beta_bar
        '''
        xi_j = delta_j - np.matmul(self.Xmat, beta_bar)

        '''
        step 4: uncover cost-side parameters and cost shocks
        '''
        alpha = beta_bar[-1] # the last coefficient in beta_bar is the price coefficie

        '''
        step -- supply side
        '''
        mcmat=self.Mcmat
        supply=np.matmul(delta_j,self.Mcmat)
        supply=(10000/supply)
        return {'beta_bar': beta_bar,'xi_j':xi_j,'supply':supply}

    def GMM_obj(self, sigmas):

        '''
        step 4: interact xi_j with instruments,which include exogenous regressors (vei
        exogenous attributes) and instruments for price (sum of attributes of competir
        '''
        xi_j = self.mean_utility(sigmas)['xi_j']
        moments = np.matmul(np.transpose(self.Zmat_D), xi_j) # an array with the shape

        '''
        step 5: compute the GMM objective function
        '''
        f = np.matmul(moments, self.weight_mat_D)
        f = (1/len(self.df)) * np.matmul(f, moments)
        return f

    def optimization(self, objfun, para):
        '''
```

```
        Parameters
        ----------
        objfun : a user defined objective function of para

        para : a 1-D array with the shape (k,), where k is the number of parameters.
        Returns
        -------
        dict
            A dictionary containing estimation results
        '''
        v = opt.minimize(objfun, x0=para, jac=None, method='BFGS',
                          options={'maxiter': 1000, 'disp': True})
        return {'obj':v.fun, "Coefficients": v.x}

if __name__ == "__main__":

    blp = BLP("iCloudDrive/Documents/Econs 514 (Metrics IV)/Assignment2/", "BLP_data.m
    pini = np.ones(len(blp.attributes_random)) * 0.2
    x = blp.GMM_obj(pini)
    beta_ols = blp.ols()
    beta_iv = blp.iv()
    beta_hedonic= blp.hedonic_price()
    print(beta_ols.summary())
    print(beta_iv.summary())
    #print(sm.OLS(blp.df['price'], blp.Zmat_D).fit().summary()) # first-stage regressi
    print(beta_hedonic.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.387
Model:                            OLS   Adj. R-squared:                  0.386
Method:                 Least Squares   F-statistic:                     279.2
Date:                Fri, 17 Mar 2023   Prob (F-statistic):         6.52e-232
Time:                        20:06:36   Log-Likelihood:                -3319.4
No. Observations:                2217   AIC:                             6651.
Df Residuals:                    2211   BIC:                             6685.
Df Model:                           5                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -10.0716      0.253    -39.822      0.000     -10.568      -9.576
x1            -0.1243      0.277     -0.448      0.654      -0.668       0.419
x2            -0.0343      0.073     -0.472      0.637      -0.177       0.108
x3             0.2650      0.043      6.146      0.000       0.180       0.350
x4             2.3421      0.125     18.707      0.000       2.097       2.588
x5            -0.0886      0.004    -22.014      0.000      -0.097      -0.081
==============================================================================
Omnibus:                      158.000   Durbin-Watson:                   1.435
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              205.595
Skew:                          -0.632   Prob(JB):                     2.27e-45
Kurtosis:                       3.792   Cond. No.                         203.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
                          IV2SLS Regression Results                           
==============================================================================
Dep. Variable:                      y   R-squared:                       0.224
Model:                         IV2SLS   Adj. R-squared:                  0.222
Method:                     Two Stage   F-statistic:                     151.3
                        Least Squares   Prob (F-statistic):         1.84e-138
Date:                Fri, 17 Mar 2023                                         
Time:                        20:06:36                                         
No. Observations:                2217                                         
Df Residuals:                    2211                                         
Df Model:                           5                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -9.7475      0.302    -32.269      0.000     -10.340      -9.155
x1             2.6760      0.930      2.878      0.004       0.853       4.499
x2             1.0455      0.348      3.009      0.003       0.364       1.727
x3             0.0712      0.078      0.917      0.359      -0.081       0.223
x4             2.2374      0.145     15.471      0.000       1.954       2.521
x5            -0.1863      0.031     -6.035      0.000      -0.247      -0.126
==============================================================================
Omnibus:                       86.419   Durbin-Watson:                   1.330
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              257.658
Skew:                          -0.034   Prob(JB):                     1.12e-56
Kurtosis:                       4.669   Cond. No.                         203.
==============================================================================
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.656
Model:                            OLS   Adj. R-squared:                  0.656
```

```
Method:                 Least Squares   F-statistic:                     844.9
Date:                Fri, 17 Mar 2023   Prob (F-statistic):               0.00
Time:                        20:06:36   Log-Likelihood:                -567.01
No. Observations:                2217   AIC:                             1146.
Df Residuals:                    2211   BIC:                             1180.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.6184      0.149     17.563      0.000       2.326       2.911
x1            -0.4706      0.049     -9.694      0.000      -0.566      -0.375
x2             0.6798      0.019     36.247      0.000       0.643       0.717
x3             0.1248      0.063      1.967      0.049       0.000       0.249
x4             0.5203      0.035     14.833      0.000       0.452       0.589
x5             0.0128      0.002      8.526      0.000       0.010       0.016
==============================================================================
Omnibus:                      533.211   Durbin-Watson:                   1.072
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1412.614
Skew:                           1.270   Prob(JB):                    1.80e-307
Kurtosis:                       5.974   Cond. No.                         307.
==============================================================================

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

In [76]:

```
v = blp.mean_utility(pini)
#v['beta_bar']
print(v)
```

```
{'beta_bar': array([-9.63171633e+00,  2.39648658e+00,  9.32968455e-01,  8.79582148e-0
3,
        2.17535992e+00, -1.77231277e-01]), 'xi_j': array([-2.74098189, -0.87860553, -
0.10204113, ..., -0.09706236,
       -1.44470651, -2.29920172]), 'supply': array([ -0.5875376 ,  -0.34473779,  -2.1
958145 ,  -0.47108033,
       -10.92093631,  -0.05480385])}
```

In [79]:

```
#pip install pyblp
import pyblp

pyblp.options.digits = 2
pyblp.options.verbose = False
pyblp.__version__
```

Out[79]:

```
'0.13.0'
```

In [80]:

```
product_data = pd.read_csv(pyblp.data.BLP_PRODUCTS_LOCATION)
product_data.head()
```

Out[80]:

	market_ids	clustering_ids	car_ids	firm_ids	region	shares	prices	hpwt	air	mpd	...
0	1971	AMGREM71	129	15	US	0.001051	4.935802	0.528997	0	1.888146	...
1	1971	AMHORN71	130	15	US	0.000670	5.516049	0.494324	0	1.935989	...
2	1971	AMJAVL71	132	15	US	0.000341	7.108642	0.467613	0	1.716799	...
3	1971	AMMATA71	134	15	US	0.000522	6.839506	0.426540	0	1.687871	...
4	1971	AMAMBS71	136	15	US	0.000442	8.928395	0.452489	0	1.504286	...

5 rows × 33 columns

In [81]:
```
agent_data = pd.read_csv(pyblp.data.BLP_AGENTS_LOCATION)
agent_data.head()
```

Out[81]:

	market_ids	weights	nodes0	nodes1	nodes2	nodes3	nodes4	income
0	1971	0.000543	1.192188	0.478777	0.980830	-0.824410	2.473301	109.560369
1	1971	0.000723	1.497074	-2.026204	-1.741316	1.412568	-0.747468	45.457314
2	1971	0.000544	1.438081	0.813280	-1.749974	-1.203509	0.049558	127.146548
3	1971	0.000701	1.768655	-0.177453	0.286602	0.391517	0.683669	22.604045
4	1971	0.000549	0.849970	-0.135337	0.735920	1.036247	-1.143436	170.226032

In [82]:
```
product_formulations = (
   pyblp.Formulation('1 + hpwt + air + mpd + space'),
   pyblp.Formulation('1 + prices + hpwt + air + mpd + space'),
   pyblp.Formulation('1 + log(hpwt) + air + log(mpg) + log(space) + trend')
)
product_formulations

agent_formulation = pyblp.Formulation('0 + I(1 / income)')
agent_formulation

problem = pyblp.Problem(product_formulations, product_data, agent_formulation, agent_c
problem

initial_sigma = np.diag([3.612, 0, 4.628, 1.818, 1.050, 2.056])
initial_pi = np.c_[[0, -43.501, 0, 0, 0, 0]]
```

```
Integration weights in the following markets sum to a value that differs from 1 by mo
re than options.weights_tol: all markets. Sometimes this is fine, for example when we
ights were built with importance sampling. Otherwise, it is a sign that there is a da
ta problem.
```

In [83]:
```
results = problem.solve(
    initial_sigma,
    initial_pi,
    costs_bounds=(0.001, None),
    W_type='clustered',
    se_type='clustered',
    initial_update=True,
```

```
)
results
```

Out[83]:
```
Problem Results Summary:
========================================================================================
==================================
GMM   Objective    Projected    Reduced Hessian  Reduced Hessian  Clipped  Clipped  W
eighting Matrix  Covariance Matrix
Step    Value    Gradient Norm  Min Eigenvalue   Max Eigenvalue   Shares    Costs   C
ondition Number  Condition Number
----  ---------  -------------  ---------------  ---------------  -------  -------  -
---------------  -----------------
 2    +5.0E+02     +6.4E-06       +4.8E-01         +5.1E+02         0        0
+4.2E+09          +3.8E+08
========================================================================================
==================================

Cumulative Statistics:
===========================================================================
Computation  Optimizer  Optimization   Objective   Fixed Point  Contraction
   Time      Converged   Iterations   Evaluations  Iterations   Evaluations
-----------  ---------  ------------  -----------  -----------  -----------
 00:08:10       No           58           159         47024       144196
===========================================================================

Nonlinear Coefficient Estimates (Robust SEs Adjusted for 999 Clusters in Parenthese
s):
========================================================================================
==============
Sigma:      1        prices      hpwt        air         mpd        space     |  Pi:
1/income
------  ----------  --------  ----------  ----------  ----------  ----------  |  ----
--  ----------
  1      +2.0E+00                                                             |    1
+0.0E+00
        (+6.1E+00)                                                            |
                                                                              |
prices   +0.0E+00   +0.0E+00                                                  |  pric
es   -4.5E+01
                                                                              |
(+9.2E+00)
                                                                              |
 hpwt    +0.0E+00   +0.0E+00   +6.1E+00                                       |   hpw
t    +0.0E+00
                              (+2.2E+00)                                      |
                                                                              |
 air     +0.0E+00   +0.0E+00   +0.0E+00    +4.0E+00                           |   air
+0.0E+00
                                          (+2.1E+00)                          |
                                                                              |
 mpd     +0.0E+00   +0.0E+00   +0.0E+00    +0.0E+00    +2.5E-01               |   mpd
+0.0E+00
                                                      (+5.5E-01)              |
                                                                              |
space    +0.0E+00   +0.0E+00   +0.0E+00    +0.0E+00    +0.0E+00    +1.9E+00   |  spac
e    +0.0E+00
                                                                  (+1.1E+00)  |
========================================================================================
==============

Beta Estimates (Robust SEs Adjusted for 999 Clusters in Parentheses):
==========================================================
    1          hpwt        air         mpd        space
```

```
----------  ----------  ----------  ----------  ----------
 -7.3E+00    +3.5E+00    -1.0E+00    +4.2E-01    +4.2E+00
(+2.8E+00)  (+1.4E+00)  (+2.1E+00)  (+2.5E-01)  (+6.6E-01)
==========================================================

Gamma Estimates (Robust SEs Adjusted for 999 Clusters in Parentheses):
======================================================================
    1       log(hpwt)      air       log(mpg)   log(space)    trend
----------  ----------  ----------  ----------  ----------  ----------
 +2.8E+00    +9.0E-01    +4.2E-01    -5.2E-01    -2.6E-01    +2.7E-02
(+1.2E-01)  (+7.2E-02)  (+8.7E-02)  (+7.3E-02)  (+2.1E-01)  (+3.1E-03)
======================================================================
```

In [ ]: