Empirical Industrial Organisations I: PSet 1

S M Sajid Al Sanai

November 2, 2018

Contents

1 Section I

1.1 Question 1

1.1.1 (a) Case for 3 Products across 100 Markets

Using the simulated data for the 3 product and 100 markets case calculated at the true parameters $\theta = \{\alpha, \beta, \sigma_\alpha\}$ and γ, I obtained the distributions for Prices, Profits, and the Consumer Surplus detailed below. Graphs for products 1-3 are aligned side-by-side for comparison.

Distribution of prices for product 1 exhibit some bimodality and is centred around USD 3 - USD 4. Distribution of prices for product 2 appears skewed slightly toward lower prices and seems centred around USD 2. Distribution of prices for product 3 appears more normally distributed around average prices of USD 3. Profits are distributed relatively similarly across products with products 2 and 3 earning some higher profits than product 1. Consumer Surplus exhibits somewhat of a bimodal distribution, with a centred mean.

Figure 1: Distribution of Prices

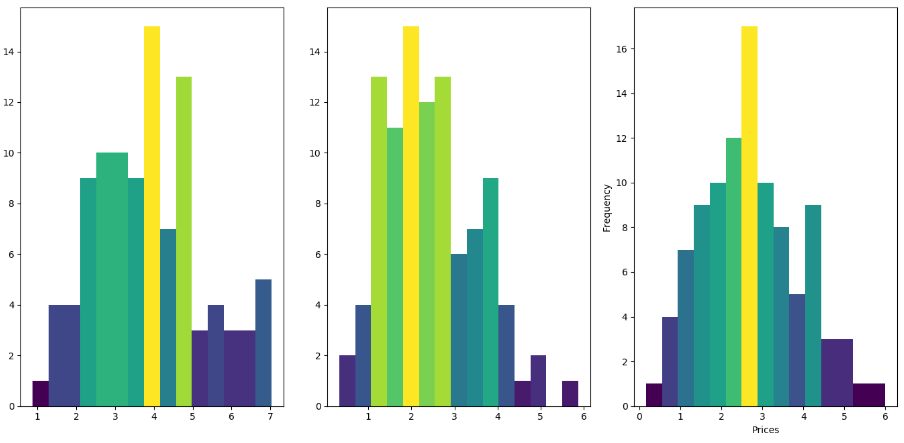


Figure 2: Distribution of Profits

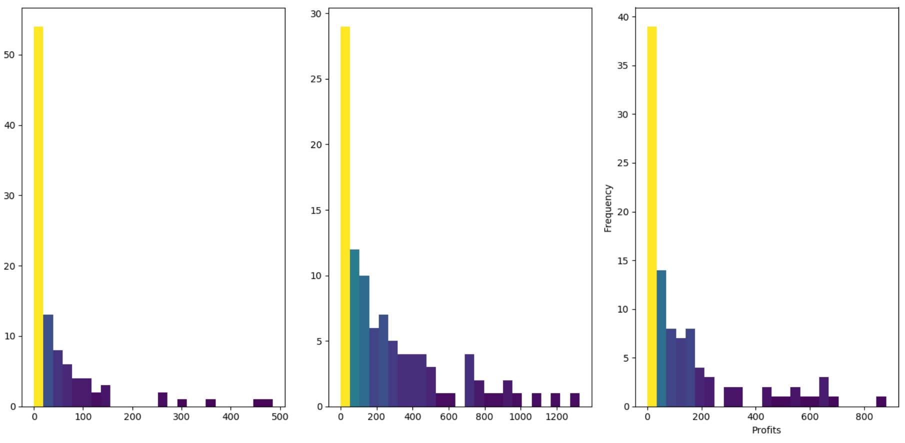
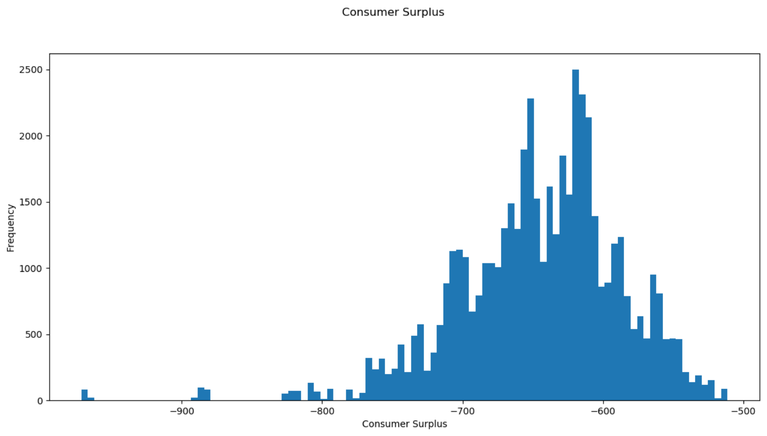


Figure 3: Distribution of Consumer Surplus



1.1.2 (b) Case for 5 Products across 100 Markets

Using the simulated data for the 5 product and 100 markets case calculated at the true parameters $\theta = \{\alpha, \beta, \sigma_\alpha\}$ and γ, I obtained the distributions for Prices, Profits, and the Consumer Surplus detailed below. Graphs for products 1-5 are aligned side-by-side for comparison.

Distribution of prices for products 1, 3, and 5 exhibit normal distribution with centring at an average price in each instance. Product 2 exhibits some bimodality. Distribution of prices for product 4 are largely skewed to negligible low average prices. Profits are distributed relatively similarly across products 1-3, and 4-5. Products 4 and 5 earning considerably higher profits than products 1-3. Consumer Surplus exhibits a normal distribution with flat tails and a centred mean.

Figure 4: Distribution of Prices

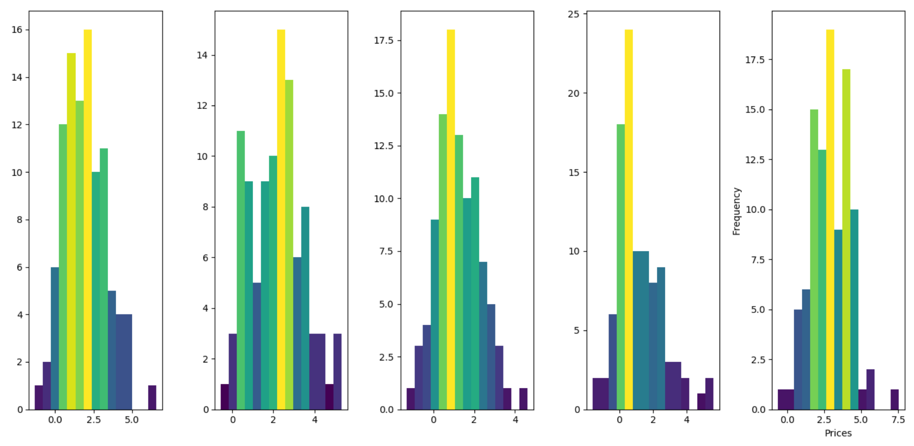


Figure 5: Distribution of Profits

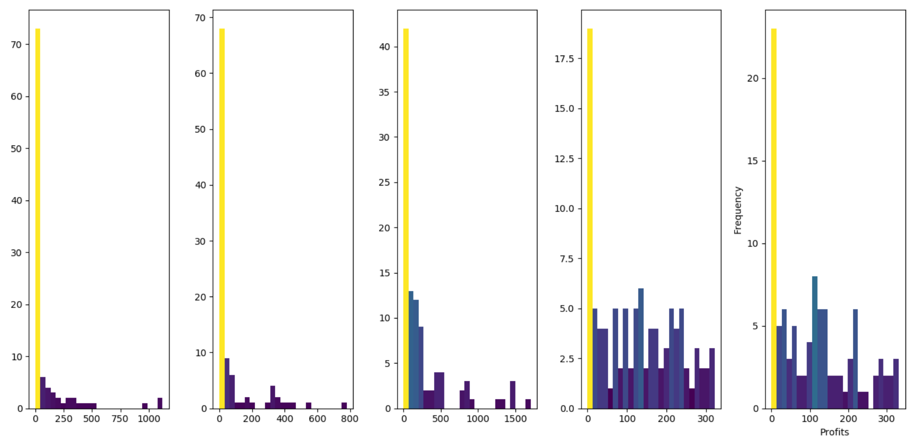
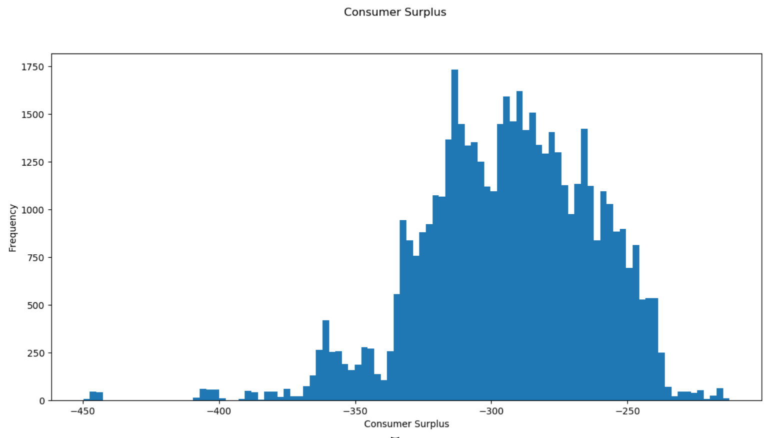


Figure 6: Distribution of Consumer Surplus



2 Section II

In answering questions from this section, I assumed that $J = 3$ and $M = 100$, except in the case of comparison to $M = 10$.

2.1 Question 1

2.1.1 (a) Computation of Moments

The value of computed moments are listed below, given assumptions $E(\xi|X) = 0$ and $E(\xi|p) = 0$.

$$E(\xi_{jm}X_{jm}) = 0.09973446265023281$$
$$E(\xi_{jm}p_{jm}) = 44.46994756$$
$$E(\xi_{jm}\bar{p}_{jm}) = 0.33463495$$

2.1.2 (b) Validity of Moments

The valid moments in this instance would be $E(\xi_{jm}X_{jm})$ and possibly $E(\xi_{jm}\bar{p}_{jm})$ given their proximity to 0. $E(\xi_{jm}p_{jm})$ would not be valid given price endogeneity, necessitating instrument variable approach.

2.1.3 (c) Use of BLP and Nevo Instruments

Given observed sample moments it would be possible to use both BLP and Nevo instruments in the instrument variables approach.

2.2 Question 2

2.2.1 (a) Computation of Moments

The BLP moment is as follows.

$$X_1 = (X', P')$$
$$\theta_1 = (\beta, \alpha)$$
$$E(\xi_{jm}Z) = (\delta - X_1\theta_1)'Z = 0$$

2.2.2 (b) Construction of Objective Function

Given that my instrument variables are denoted by Z and weighting matrix of A, and my parameters are specified as $\theta_1 = \{\beta, \alpha\}$ and $\theta_2 = \{\sigma_\alpha\}$, I used a contraction mapping procedure to estimate mean utility δ over numerous iterations given a tolerance until convergence. My instrument variable Z is constructed with the inclusion of X and non-self product characteristics.

Using this estimated mean utility, I constructed my objective function and minimised over θ_2 only by non-linear search, while re-estimating θ_1 using a first order condition within the loop. X is a vector including characteristics and prices across all markets. My initial guess for δ_0 were the given market shares, which were used to obtain initial starting values for parameters θ_1 for subsequent iterations.

$$A = Z'Z$$
$$\omega(\theta_1) = \delta - X_1\theta_1$$
$$\hat{\theta_1} = (X_1'Z(A)^{-1}Z'X_1)^{-1}(X_1'Z(A)^{-1}Z'\delta(\hat{\theta_2}))$$
$$\hat{\theta_2} = \underset{\theta}{argmin}\,(\omega(\hat{\theta_1})'Z)(A)^{-1}(Z'\omega(\hat{\theta_1}))$$

2.2.3 (c) Estimation of θ Parameters

An error with my prediction of market shares function has led to the generation of poor estimates for θ_1 given prior knowledge of the true distribution of each variable.

$$\hat{\beta_0} = 25.36425337\ (0.1156)$$
$$\hat{\beta_1} = -1.48315299\ (3.0040)$$
$$\hat{\beta_2} = 0.18493143\ (0.0401)$$
$$\hat{\alpha} = -8.37344604\ (0.0036)$$
$$\hat{\sigma_\alpha} = 1.075\ (1.2652)$$

We observe statistical insignificance in all estimators with the exception of $\hat{\beta_2}$ and $\hat{\alpha}$. Code output reporting variances are as follows.

```
Estimation of Parameters and respective Standard Errors:
Beta[0]:      [25.36425337]; (0.11561192456479381) 0.013366117101575576
Beta[1]:      [-1.48315299]; (3.004018606720816) 9.024127789524872
Beta[2]:      [0.18493143]; (0.04011846630681128) 0.001609491338810752
Alpha:        [-8.37344604]; (0.003586413934865264) 1.2862364912195745e-05
Sigma Alpha:  [1.075]; (1.265210529517815) 1.6007576840027502
```

2.2.4 (d) Estimated Distributions

The distribution of profits do not change greatly for each product from generating the same figures from the dataset using the true parameters. The distribution of the Consumer Surplus is smoother given our estimated parameters with a slightly flattened peak around the centred mean.

Figure 7: Distribution of Profits

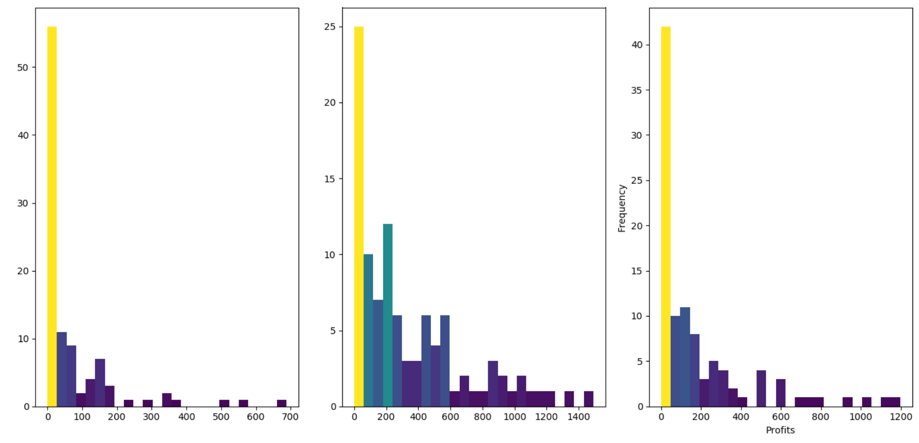
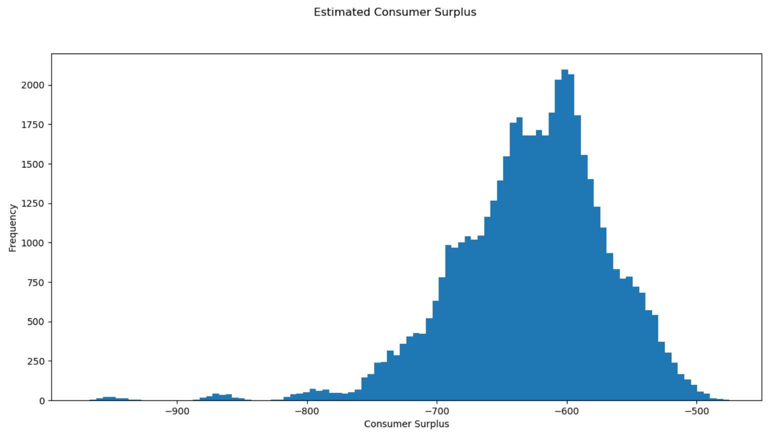


Figure 8: Distribution of Consumer Surplus



2.2.5 (e) Estimation with Small Market Size

The following assumes that $J = 3$ and $M = 10$. Estimated parameters are as follows.

$$\hat{\beta}_0 = 4.93649959\ (0.5500)$$

$$\hat{\beta}_1 = -6.26911567\ (3.6350)$$

$$\hat{\beta}_2 = 0.64225405\ (2.7504)$$

$$\hat{\alpha} = -3.05454773\ (0.0680)$$

$$\hat{\sigma_\alpha} = 1.05\ (1.6873)$$

We observe statistical insignificance in all estimators. Unusually, estimates come closer to true parameter values given the smaller market size. This is again due to aforementioned inadequacies in the share prediction function with which the contraction mapping is iterated to determine mean utility. Predictably, given large sample asymptotics, standard errors for each parameter grow higher as the size of the sample is smaller, implying a lack of precision. Code output reporting variances are as follows.

```
Estimation of Parameters and respective Standard Errors:
Beta[0]:      [4.93649959]; (0.5500157661303665) 0.302517342991974
Beta[1]:      [-6.26911567]; (3.634987444230612) 13.213133719714198
Beta[2]:      [0.64225405]; (2.7504394588441516) 7.5649172167669105
Alpha:        [-3.05454773]; (0.06798615060671363) 0.004622116674318748
Sigma Alpha: [1.05]; (1.6872506698774878) 2.846814823002031
```

Figure 9: Distribution of Profits

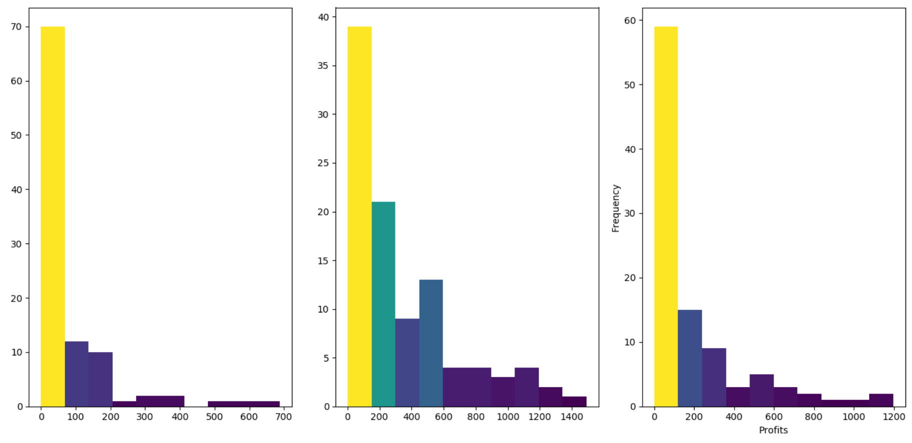
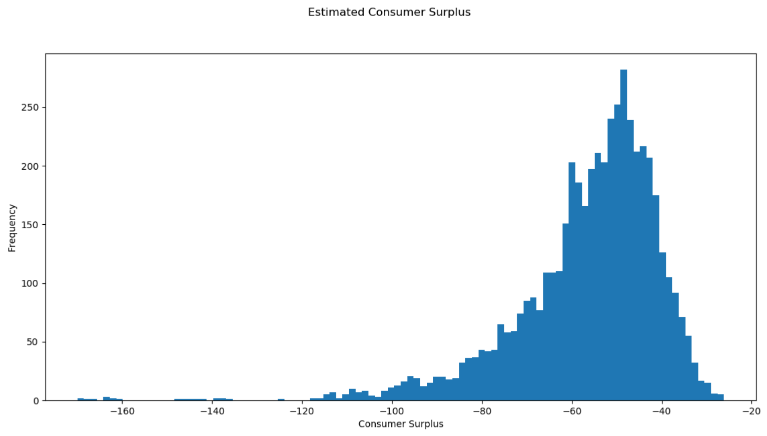


Figure 10: Distribution of Consumer Surplus



3 Section III

3.1 Question 1

3.1.1 (a) Computation of Moments

$$X_1 = (X', P')$$

$$\theta_1 = (\beta, \alpha)$$

$$Z = (X', W')$$

$$E(\xi_{jm} Z) = (\delta - X_1\theta_1)' Z = 0$$

3.1.2 (b) Estimation of θ Parameters

Given that my instrument variables are denoted by Z and weighting matrix of A, and my parameters are specified as $\theta_1 = \{\beta, \alpha\}$ and $\theta_2 = \{\sigma_\alpha\}$, I used a contraction mapping procedure to estimate mean utility δ over numerous iterations given a tolerance until convergence. The key difference here is that my instrument variable Z is constructed with the inclusion of X, W (supply side), and non-self product characteristics.

Using this estimated mean utility, I constructed my objective function and minimised over θ_2 only by non-linear search, while re-estimating θ_1 using a first order condition within the loop. X is a vector including characteristics and prices across all markets. My initial guess for δ_0 were the given market shares, which were used to obtain initial starting values for parameters θ_1 for subsequent iterations.

$$\hat{\beta_0} = 3.44581615\,(1.7764)$$

$$\hat{\beta_1} = -0.26703854\,(2.1939)$$

$$\hat{\beta_2} = -0.07335316\,(0.0440)$$

$$\hat{\alpha} = -1.47744888\,(0.0819)$$

$$\hat{\sigma_\alpha} = -0.025\,(0.9368)$$

We observe statistical insignificance in all estimators with the exception of $\hat{\beta_2}$.

```
Estimation of Parameters and respective Standard Errors:
Beta[0]:      [3.44581615]; (1.7763856722429023) 3.155546056549868
Beta[1]:      [-0.26703854]; (2.1939101073905936) 4.8132415593106055
Beta[2]:      [-0.07335316]; (0.044034215343053545) 0.0019390121208784123
Alpha:        [-1.47744888]; (0.08189215439712716) 0.006706324951802914
Sigma Alpha:  [-0.025]; (0.936762973259827) 0.8775248680705915
```

Figure 11: Distribution of Profits

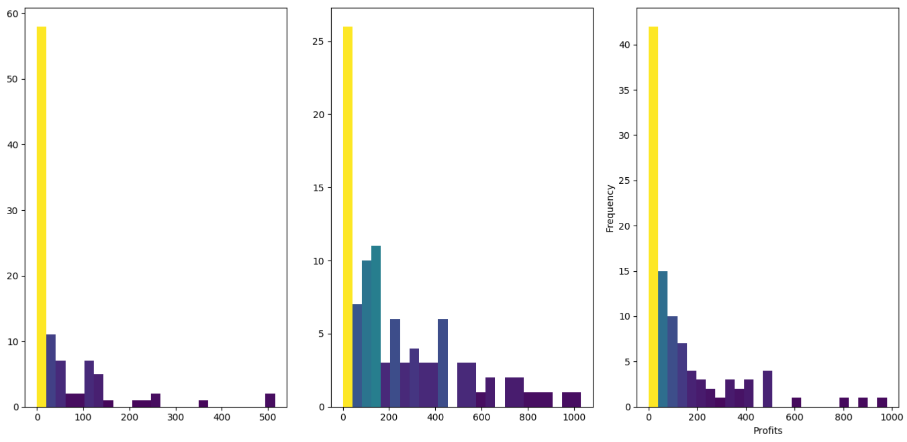
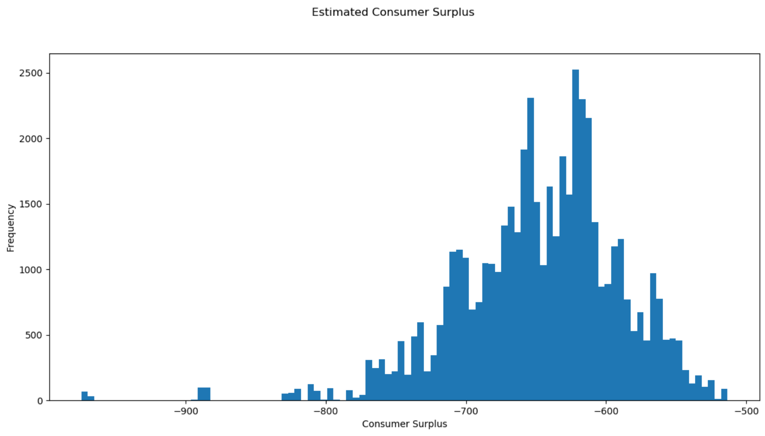


Figure 12: Distribution of Consumer Surplus



3.1.3 (c) Estimation with Small Market Size

The following assumes that $J = 3$ and $M = 10$. Estimated parameters are as follows.

$$\hat{\beta_0} = 7.200781\ (1.4963)$$

$$\hat{\beta_1} = -3.14960642\ (14.1955)$$

$$\hat{\beta_2} = 0.67499153\ (5.8012)$$

$$\hat{\alpha} = -4.10794263\ (0.0266)$$

$$\hat{\sigma_\alpha} = 1.\ (6.6877)$$

Expectedly, estimates stray farther than true parameter values and also in comparison to prior estimation, given the smaller market size. This is again due to aforementioned inadequacies in the share prediction function with which the contraction mapping is iterated to determine mean utility. Predictably, given large sample asymptotics, standard errors for each parameter grow higher as the size of the sample is smaller, implying a lack of precision.

Statistical insignificance persists overall, however $\hat{\alpha}$ appears to be statistically significant in this estimation compared to the case with 100 markets. Distributions follow closely, however, Consumer Surplus is considerably smoother without twin peaks in the case of the smaller market. Code output reporting variances are as follows.

```
Estimation of Parameters and respective Standard Errors:
Beta[0]:       [7.200781]; (1.4962813893043092) 2.238857995978434
Beta[1]:       [-3.14960642]; (14.195538337691184) 201.51330869686018
Beta[2]:       [0.67499153]; (5.8012724739414185) 33.654762316910386
Alpha:         [-4.10794263]; (0.026634193525163823) 0.0007093802647358785
Sigma Alpha:  [1.]; (6.687712926310157) 44.72550418473597
```

Figure 13: Distribution of Profits

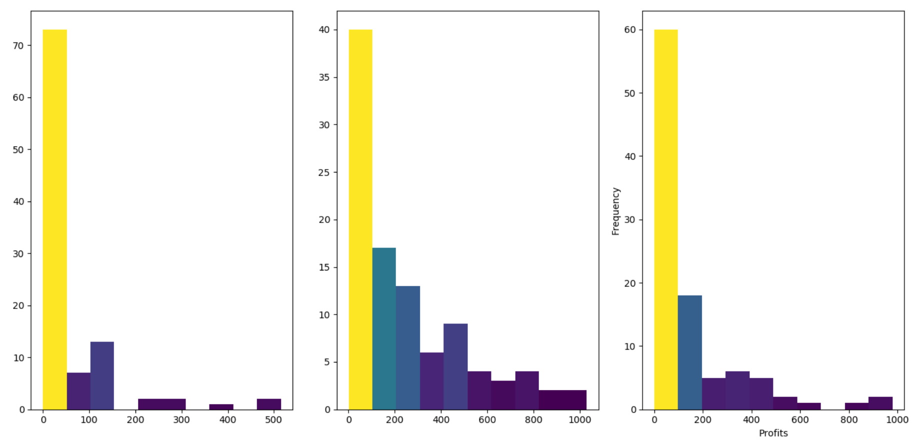
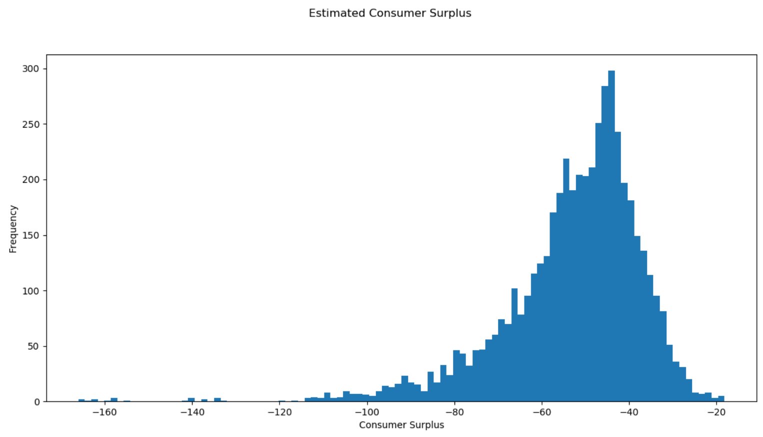


Figure 14: Distribution of Consumer Surplus



4 Appendix: Source Code

4.1 Source

```
# Import Libraries
import numpy as np;
import scipy as sp;
from scipy import optimize;
from scipy import io;
import pandas as pd;
import matplotlib.pyplot as plt;
from matplotlib import colors;
```

```
# Section 1:
print( 'Section 1:' )
print( 'Question 1:' )
print('')

def f_ObtainConsumerSurplus( X, P, Xi, alpha, beta, estimated ):
    Mu      = np.zeros( shape=(num_products, num_markets, num_consumers) )
    welfare = np.zeros( shape=(num_markets, num_consumers) )

    X_0 = np.reshape( X[:, 0], (num_markets, num_products) ).T
    X_1 = np.reshape( X[:, 1], (num_markets, num_products) ).T
    X_2 = np.reshape( X[:, 2], (num_markets, num_products) ).T

    for j in range(num_products):
        Mu[j, :, :] = -np.matmul( np.reshape( P[j].T, (num_markets, 1) ).T, alpha )
    for j in range(num_products):
        Mu[j, :, :] += np.reshape( ( X_0 * beta[0] + X_1 * beta[1] + X_2 * beta[2] + np
    for m in range(num_markets):
        for i in range(num_consumers):
            welfare[m, i] = np.max( Mu[:, m, i] )

    fig = plt.figure();
    ax = fig.add_subplot( 1, 1, 1 );
    ax.hist( np.reshape( welfare, (num_consumers*num_markets, 1) ), bins=100);
    ax.set_xlabel('Consumer Surplus');
    ax.set_ylabel('Frequency');
    plt.subplots_adjust(wspace=0.4, hspace=0.4);

    if estimated == True:
       fig.suptitle('Estimated Consumer Surplus');
    else:
       fig.suptitle('Consumer Surplus');

    plt.show()
    print('')

    return True

# Import Dataset
dataset_file = '100markets3products.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

print( sp.io.whosmat( dataset_file ) )
print('')

# Dataset Characteristics
num_markets   = 100
num_products  = 3
num_consumers = 500
```

```
print( 'Markets:    ' + str( num_markets ) )
print( 'Products:   ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Generating New ndarray Variables from Dataset
x1_3     = np.array( dataset_raw['x1'] )
xi_all_3 = np.array( dataset_raw['xi_all'] )
w_3      = np.array( dataset_raw['w'] )
eta_3    = np.array( dataset_raw['eta'] )
Z_3      = np.array( dataset_raw['Z'] )
alphas_3 = np.array( dataset_raw['alphas'] )
P_opt_3  = np.array( dataset_raw['P_opt'] )
shares_3 = np.array( dataset_raw['shares'] )

print( 'x1      ' + str(x1_3.shape) )
print( 'xi_all  ' + str(xi_all_3.shape) )
print( 'w       ' + str(w_3.shape) )
print( 'eta     ' + str(eta_3.shape) )
print( 'Z       ' + str(Z_3.shape) )
print( 'alphas  ' + str(alphas_3.shape) )
print( 'P_opt   ' + str(P_opt_3.shape) )
print( 'shares  ' + str(shares_3.shape) )
print('')

price_product1_3 = P_opt_3[0,:]
price_product2_3 = P_opt_3[1,:]
price_product3_3 = P_opt_3[2,:]
price_product1_3 = np.reshape( price_product1_3, (-1, 1) )
price_product2_3 = np.reshape( price_product2_3, (-1, 1) )
price_product3_3 = np.reshape( price_product3_3, (-1, 1) )
share_product1_3 = shares_3[0,:]
share_product2_3 = shares_3[1,:]
share_product3_3 = shares_3[2,:]
share_product1_3 = np.reshape( share_product1_3, (-1, 1) )
share_product2_3 = np.reshape( share_product2_3, (-1, 1) )
share_product3_3 = np.reshape( share_product3_3, (-1, 1) )

# Generating Consumer Surplus
f_ObtainConsumerSurplus( x1_3, P_opt_3, xi_all_3, alphas_3, np.reshape( [5, 1, 1], (3,

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_3 = np.zeros( shape=( num_markets * num_products, 1 ) )
for i in range(num_markets * num_products):
    mc_3[i] = gamma[0] * 1 + gamma[1] * w_3[i] + gamma[2] * Z_3[i] + eta_3[i]

# Generating Profits
profit_product1_3 = np.zeros( shape=( num_markets, 1 ) )
profit_product2_3 = np.zeros( shape=( num_markets, 1 ) )
```

```
profit_product3_3 = np.zeros( shape=( num_markets, 1 ) )
for i in range( num_markets ):
    profit_product1_3[i] = ( price_product1_3[i] - mc_3[i*num_products] ) * num_consum
    profit_product2_3[i] = ( price_product2_3[i] - mc_3[i*num_products+1] ) * num_consu
    profit_product3_3[i] = ( price_product3_3[i] - mc_3[i*num_products+2] ) * num_consu

# Generate Histogram
print( 'Generating Histogram of Distribution for Prices ...' )
num_bins = 15
fig, histogram = plt.subplots(1, 3, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(price_product1_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(price_product2_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(price_product3_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
plt.show()

print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 25
fig, histogram = plt.subplots(1, 3, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(profit_product1_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(profit_product2_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
```

```
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(profit_product3_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')

# Import Dataset
dataset_file = '100markets5products.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

print( sp.io.whosmat( dataset_file ) )
print('')

# Dataset Characteristics
num_markets   = 100
num_products  = 5
num_consumers = 500

print( 'Markets:   ' + str( num_markets ) )
print( 'Products:  ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Generating New ndarray Variables from Dataset
x1_5     = np.array( dataset_raw['x1'] )
xi_all_5 = np.array( dataset_raw['xi_all'] )
w_5      = np.array( dataset_raw['w'] )
eta_5    = np.array( dataset_raw['eta'] )
Z_5      = np.array( dataset_raw['Z'] )
alphas_5 = np.array( dataset_raw['alphas'] )
P_opt_5  = np.array( dataset_raw['P_opt'] )
shares_5 = np.array( dataset_raw['shares'] )

print( 'x1      ' + str(x1_5.shape) )
print( 'xi_all  ' + str(xi_all_5.shape) )
print( 'w       ' + str(w_5.shape) )
print( 'eta     ' + str(eta_5.shape) )
print( 'Z       ' + str(Z_5.shape) )
print( 'alphas  ' + str(alphas_5.shape) )
print( 'P_opt   ' + str(P_opt_5.shape) )
print( 'shares ' + str(shares_5.shape) )
print('')

price_product1_5 = P_opt_5[0,:]
price_product2_5 = P_opt_5[1,:]
price_product3_5 = P_opt_5[2,:]
price_product4_5 = P_opt_5[3,:]
price_product5_5 = P_opt_5[4,:]
price_product1_5 = np.reshape( price_product1_5, (-1, 1) )
price_product2_5 = np.reshape( price_product2_5, (-1, 1) )
price_product3_5 = np.reshape( price_product3_5, (-1, 1) )
price_product4_5 = np.reshape( price_product4_5, (-1, 1) )
price_product5_5 = np.reshape( price_product5_5, (-1, 1) )
share_product1_5 = shares_5[0,:]
share_product2_5 = shares_5[1,:]
share_product3_5 = shares_5[2,:]
share_product4_5 = shares_5[2,:]
share_product5_5 = shares_5[2,:]
share_product1_5 = np.reshape( share_product1_5, (-1, 1) )
share_product2_5 = np.reshape( share_product2_5, (-1, 1) )
share_product3_5 = np.reshape( share_product3_5, (-1, 1) )
share_product4_5 = np.reshape( share_product4_5, (-1, 1) )
share_product5_5 = np.reshape( share_product5_5, (-1, 1) )

# Generating Consumer Surplus
f_ObtainConsumerSurplus( x1_5, P_opt_5, xi_all_5, alphas_5, np.reshape( [5, 1, 1], (3,

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_5 = np.empty( shape=( num_markets * num_products, 1 ) )
for i in range(num_markets * num_products):
    mc_5[i] = gamma[0] * 1 + gamma[1] * w_5[i] + gamma[2] * Z_5[i] + eta_5[i]

# Generating Profits
profit_product1_5 = np.zeros( shape=( num_markets, 1 ) )
profit_product2_5 = np.zeros( shape=( num_markets, 1 ) )
profit_product3_5 = np.zeros( shape=( num_markets, 1 ) )
profit_product4_5 = np.zeros( shape=( num_markets, 1 ) )
profit_product5_5 = np.zeros( shape=( num_markets, 1 ) )
for i in range( num_markets ):
    profit_product1_5[i] = ( price_product1_5[i] - mc_5[i*num_products] ) * num_consum
    profit_product2_5[i] = ( price_product2_5[i] - mc_5[i*num_products+1] ) * num_consu
    profit_product3_5[i] = ( price_product3_5[i] - mc_5[i*num_products+2] ) * num_consu
    profit_product4_5[i] = ( price_product4_5[i] - mc_5[i*num_products+3] ) * num_consu
    profit_product5_5[i] = ( price_product5_5[i] - mc_5[i*num_products+4] ) * num_consu

# Generate Histogram
print( 'Generating Histogram of Distribution for Prices ...' )
num_bins = 15
fig, histogram = plt.subplots(1, 5, sharey=False, tight_layout=True)
```

```
N, bins, patches = histogram[0].hist(price_product1_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(price_product2_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(price_product3_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[3].hist(price_product4_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
N, bins, patches = histogram[4].hist(price_product5_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Prices');
plt.ylabel('Frequency');
plt.show()

print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 25
fig, histogram = plt.subplots(1, 5, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(profit_product1_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(profit_product2_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(profit_product3_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[3].hist(profit_product4_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[4].hist(profit_product5_5, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')
print('')


# Section 2
print( 'Section 2' )
print( 'Question 1' )
print('')

# Dataset Characteristics
num_markets   = 100
num_products  = 3
num_consumers = 500
```

```
print( '[ E(Xi*X) ]' )
E_xi_x_3 = np.array( [np.matmul( xi_all_3.T, x1_3[:,0] ), np.matmul( xi_all_3.T, x1_3[:
for i in range(3):
    E_xi_x_3[0,i] = ( 1 / (num_markets * num_products) ) * E_xi_x_3[0,i]
print( str( E_xi_x_3.shape ) + ': ' + str( E_xi_x_3 ) + ', ' + str( np.sum( E_xi_x_3 )
print('')

print( '[ E(Xi*P) ]' )
E_xi_p_3 = ( np.reshape( P_opt_3, (300, 1) ).T.dot(xi_all_3) )
print( str( E_xi_p_3.shape ) + ': ' + str( E_xi_p_3 ) )
print('')

print( '[ E(Xi*P-bar) ]' )
pbar_3 = np.zeros( shape=(num_products, num_markets) )
a = pd.DataFrame(P_opt_3)
b = np.zeros( shape=(3, 100) )
for j in range(num_markets):
    for i in range(num_products):
        b = a.drop( columns = j )
        pbar_3[i, j] = np.mean( b.values[i,:] )
pbar_3 = np.reshape( pbar_3, (300, 1) )
E_xi_pbar_3 = ( 1 / num_markets ) * xi_all_3.T.dot(pbar_3)
print( str( E_xi_pbar_3.shape ) + ': ' + str( E_xi_pbar_3 ) )
print('')


print( 'Question 2' )
print('')

### ****** Function Definitions
def f_ContractionMapping( initial_mean_utility, sigma_alpha, tolerance, max_iterations
    # Iterating Loop for Contraction Mapping
    print('*** Initiating Contraction Mapping ***')
    i = 0
    norm_avg = 1
    #t_delta = initial_delta
    iterated_mean_utility = initial_mean_utility
    while ( norm_avg > tolerance  and i < max_iterations ):
        initial_mean_utility, iterated_mean_utility, predicted_shares = f_ContractMean
        norm_avg = np.mean( np.abs( np.subtract( iterated_mean_utility, initial_mean_u
        print( 'Iteration Step [' + str(i) + ']' )
        i += 1
    print( '*** Complete ***' )
    print( 'Norm (Avg): ' + str( norm_avg ) )
    print( 'Iterations: ' + str( i ) )
    print( 'Returning Matrix of Market Shares of Dimension ' + str(predicted_shares.sha
    #print( predicted_shares )
    print( 'Returning Matrix of Estimated Mean Utility ' + str(iterated_mean_utility.sh
    #print( iterated_mean_utility )
    print( 'Returning Matrix of Differences ' + str( (np.subtract( iterated_mean_utilit
    #print( np.subtract( iterated_mean_utility , initial_mean_utility ) )

    return iterated_mean_utility , predicted_shares

def f_ContractMeanUtility( initial_mean_utility , sigma_alpha ):
    # Iteration
    predicted_shares      = f_PredictMarketShares( sigma_alpha , initial_mean_utility )
    if num_markets == 100:
        iterated_mean_utility = initial_mean_utility + np.log( shares_3 ) - np.log( pre
    if num_markets == 10:
        iterated_mean_utility = initial_mean_utility + np.log( shares_10 ) - np.log( p

    return initial_mean_utility , iterated_mean_utility , predicted_shares

def f_PredictMarketShares( sigma_alpha , mean_utility ):
    # Calculation of Predicted Shares
    numerator        = np.zeros( (num_products , num_consumers , num_markets) )
    denominator      = 1
    predicted_shares = np.zeros( (num_products , num_consumers , num_markets) )
    out              = np.zeros( (num_products , num_markets) )
    for j in range(num_products):
        numerator[j , :, :] = np.exp( mean_utility[j , :] + nu.T * sigma_alpha * P[j , :].
        denominator += numerator[j , :, :]
    for j in range(num_products):
        predicted_shares[j , :, :] = numerator[j , :, :] / denominator
        out[j , :] = np.mean( predicted_shares[j , :, :], axis = 0 )

    return out

def f_ObtainOLSParameters( X, Y ):
    # Closed Form Expression for Ordinary Least Squares Regression Parameter Estimates
    beta = ( np.linalg.inv(X.T.dot( X )) ).dot( X.T ).dot( Y )

    return beta

def f_ObtainMeanUtility( X, theta2 ):
    # delta_jm = X_jm * theta2[0:3] + theta2[4] * price_jm
    mean_utility = np.matmul( X, theta2 )
    mean_utility = np.reshape( np.array( mean_utility ), (num_products , num_markets), o

    return mean_utility

# Defining GMM Objective Function
def f_GMM( sigma_alpha ):
    guess_mean_utility = f_ObtainMeanUtility( guess_X , estimated_theta1 )
    # [theta1] beta , alpha; [theta2] sigma_alpha;

    print( 'Constructing Guess for Mean Utility (delta0_jm):' )
    print('')
    print( 'Beta[0]:      ' + str( estimated_theta1[0] ) )
    print( 'Beta[1]:       ' + str( estimated_theta1[1] ) )
    print( 'Beta[2]:       ' + str( estimated_theta1[2] ) )
    print( 'Alpha:         ' + str( estimated_theta1[3] ) )
    print( 'Sigma_Alpha: ' + str( sigma_alpha ) )
    print('')
    print( 'Constructed Matrix of Mean Utility (delta0_jm) ...' )
    print( guess_mean_utility.shape )
    print('')

    # Initiate Contraction Mapping
    #estimated_mean_utility, estimated_shares = f_ContractionMapping( guess_mean_utilit
    estimated_mean_utility, estimated_shares = f_ContractionMapping( guess_mean_utility

    global estimated_mean_utility_iterated
    estimated_mean_utility_iterated = estimated_mean_utility

    global estimated_market_shares
    estimated_market_shares = estimated_shares

    if np.max( np.isnan( estimated_mean_utility ) == True ):
        out = 1e+10
    else:
        # First Order Conditions
        estimated_mu = np.reshape( estimated_mean_utility, (num_products*num_markets, 
        estimated_mu = np.reshape( estimated_mean_utility, (num_products*num_markets, 
        temp1 = np.matmul( guess_X.T, IV )
        temp2 = np.matmul( estimated_mu.T, IV )
        invA  = np.linalg.inv( np.matmul( IV.T, IV ) )
        estimated_theta1[0:4] = np.matmul( np.linalg.inv( np.matmul( temp1, np.matmul( 
        gmm_residuals = np.subtract( estimated_mu, np.matmul( guess_X, estimated_theta1
        temp1 = np.matmul( gmm_residuals.T, IV )
        out = np.matmul( temp1, np.matmul( invA, temp1.T ) )

    return out

def f_ObtainInstrumentVariables( num_markets, num_products, X ):
    IV = np.ones( (num_markets*num_products, 5) )
    IV[:, 0:3] = X[:, 0:3]
    X_23 = np.reshape( X[:, 1:3], (num_markets, num_products, 2) )
    Z_45 = np.ones( (num_markets, num_products, 2 ) )
    for i in range(2):
        for j in range(num_products):
            Z_45[:, j, i] = np.mean( np.delete( X_23[:, :, i], j, axis=1 ), axis=1 )
        IV[:, 3 + i] = np.reshape( Z_45[:, :, i], (num_markets*num_products, 1) )[:, 0]

    return IV

def f_ObtainVarCov( sigma_alpha ):
    mu = f_ObtainMeanUtility( guess_X, estimated_theta1 )
    hi_mu, s = f_ContractionMapping( mu, np.add( sigma_alpha, 0.1 ), 0.5, 100 )
    md_mu, s = f_ContractionMapping( mu, sigma_alpha, 0.5, 100 )
    lo_mu, s = f_ContractionMapping( mu, np.subtract( sigma_alpha, 0.1 ), 0.5, 100 )

    hi_mu = np.reshape( hi_mu, (num_markets*num_products, 1) )
    md_mu = np.reshape( md_mu, (num_markets*num_products, 1) )
    lo_mu = np.reshape( lo_mu, (num_markets*num_products, 1) )

    gradient_mu = ( 1 / 0.2 ) * np.subtract( hi_mu, lo_mu )
    gradient = np.concatenate( (gradient_mu, -guess_X), 1)

    Q     = ( 1 / (num_markets * num_products) ) * np.matmul( IV.T, gradient )
    g_u   = np.matmul( np.subtract( md_mu, np.matmul( guess_X, estimated_theta1 ) ).T,
    invA  = np.linalg.inv( np.matmul( IV.T, IV ) )
    Omega = ( 1 / (num_markets * num_products) ) * np.matmul( g_u.T, g_u )

    out   = np.linalg.inv( np.matmul( Q.T, np.matmul( invA, Q ) ) )
    out   = np.matmul( out, np.matmul( Q.T, np.matmul( invA, np.matmul( Omega, np.matm
    out   = np.matmul( out, np.linalg.inv( np.matmul( Q.T, np.matmul( invA, Q ) ) ) )
    out   = ( 1 / (num_markets * num_products) ) * out

    return out
### ******

# Dataset Characteristics
print( 'Markets:   ' + str( num_markets ) )
print( 'Products:  ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Set Seed Value for Future Random Draws
np.random.seed( 1337 )

# Initial Setup for Guess of Mean Utility
P  = np.array( P_opt_3 )
X  = np.array( x1_3 )
X  = np.concatenate( ( X, np.reshape( P, (num_products*num_markets, 1) ) ), 1 )
Z  = np.array( np.reshape( Z_3, (num_products, num_markets), order='F' ) )

# Log Normally Distributed Draws for Nu
nu = np.zeros( shape=( num_consumers * num_markets ) )
for i in range(num_consumers * num_markets):
    nu[i] = np.random.lognormal( 0.0, 1.0 )
nu = np.reshape( nu, (num_markets, num_consumers) )

# Initial Guess for Mean Utility
guess_X = X
guess_Y = np.array( np.reshape( shares_3, (num_products*num_markets, 1) ) )

# Constructing Instruments
```

```
IV = f_ObtainInstrumentVariables( num_markets, num_products, guess_X )

# Commence Optimization of GMM Objective Function
estimated_market_shares = np.ones( (num_products, num_markets) )
estimated_mean_utility_iterated = np.ones( (num_products, num_markets) )

estimated_theta1 = f_ObtainOLSParameters( guess_X, guess_Y )
#estimated_theta1 = [1, 1, 1, 1]

# Using FMinSearch()
estimated_theta2 = sp.optimize.fmin( f_GMM, 1 )

# Using BFGS()
#estimated_theta2 = sp.optimize.minimize( f_GMM, 1, method='BFGS' )

# Using Basin-Hopping
#estimated_theta2 = sp.optimize.basinhopping( f_GMM, 1 )

# Calculating Estimated Prices and Market Shares
price_product1_3 = P[0,:]
price_product2_3 = P[1,:]
price_product3_3 = P[2,:]
price_product1_3 = np.reshape( price_product1_3, (-1, 1) )
price_product2_3 = np.reshape( price_product2_3, (-1, 1) )
price_product3_3 = np.reshape( price_product3_3, (-1, 1) )
share_product1_3 = estimated_market_shares[0,:]
share_product2_3 = estimated_market_shares[1,:]
share_product3_3 = estimated_market_shares[2,:]
share_product1_3 = np.reshape( share_product1_3, (-1, 1) )
share_product2_3 = np.reshape( share_product2_3, (-1, 1) )
share_product3_3 = np.reshape( share_product3_3, (-1, 1) )

# Variance-Covariance Matrix Determination
varcovar = np.diag( f_ObtainVarCov( estimated_theta2 ) )
stderror = np.sqrt( varcovar )

print('')
print('Estimation of Parameters and respective Standard Errors:')
print( 'Beta[0]:      ' + str( estimated_theta1[0] ) + '; (' + str( stderror[0] ) + ')
print( 'Beta[1]:      ' + str( estimated_theta1[1] ) + '; (' + str( stderror[1] ) + ')
print( 'Beta[2]:      ' + str( estimated_theta1[2] ) + '; (' + str( stderror[2] ) + ')
print( 'Alpha:        ' + str( estimated_theta1[3] ) + '; (' + str( stderror[3] ) + ')
print( 'Sigma Alpha: ' + str( estimated_theta2 ) + '; (' + str( stderror[4] ) + ') ' +
print('')

# Price Elasticities of Demand
##############################

# Generating Consumer Surplus
Xi = np.reshape( estimated_mean_utility_iterated, (num_markets*num_products, 1) ) - np.
```

```
f_ObtainConsumerSurplus( x1_3, P_opt_3, Xi, alphas_3, estimated_theta1[0:3], True )

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_3 = np.zeros( shape=( num_markets * num_products, 1 ) )
for i in range(num_markets * num_products):
    mc_3[i] = gamma[0] * 1 + gamma[1] * w_3[i] + gamma[2] * Z_3[i] + eta_3[i]

# Generating Profits
profit_product1_3 = np.zeros( shape=( num_markets, 1 ) )
profit_product2_3 = np.zeros( shape=( num_markets, 1 ) )
profit_product3_3 = np.zeros( shape=( num_markets, 1 ) )
for i in range( num_markets ):
    profit_product1_3[i] = ( price_product1_3[i] - mc_3[i*num_products] ) * num_consum
    profit_product2_3[i] = ( price_product2_3[i] - mc_3[i*num_products+1] ) * num_consu
    profit_product3_3[i] = ( price_product3_3[i] - mc_3[i*num_products+2] ) * num_consu

# Generate Histogram
print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 25
fig, histogram = plt.subplots(1, 3, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(profit_product1_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(profit_product2_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(profit_product3_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')


#num_markets = 10
```

```
###############################
# Dataset Characteristics
num_markets = 10

# Import Dataset
dataset_file = '10markets3products.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

print( sp.io.whosmat( dataset_file ) )
print('')

print( 'Markets:   ' + str( num_markets ) )
print( 'Products:  ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Generating New ndarray Variables from Dataset
x1_10     = np.array( dataset_raw['x1'] )
xi_all_10 = np.array( dataset_raw['xi_all'] )
w_10      = np.array( dataset_raw['w'] )
eta_10    = np.array( dataset_raw['eta'] )
Z_10      = np.array( dataset_raw['Z'] )
alphas_10 = np.array( dataset_raw['alphas'] )
P_opt_10  = np.array( dataset_raw['P_opt'] )
shares_10 = np.array( dataset_raw['shares'] )

print( 'x1      ' + str(x1_10.shape) )
print( 'xi_all  ' + str(xi_all_10.shape) )
print( 'w       ' + str(w_10.shape) )
print( 'eta     ' + str(eta_10.shape) )
print( 'Z       ' + str(Z_10.shape) )
print( 'alphas  ' + str(alphas_10.shape) )
print( 'P_opt   ' + str(P_opt_10.shape) )
print( 'shares  ' + str(shares_10.shape) )
print('')

# Set Seed Value for Future Random Draws
np.random.seed( 1337 )

# Initial Setup for Guess of Mean Utility
P  = np.array( P_opt_10 )
X  = np.array( x1_10 )
X  = np.concatenate( ( X, np.reshape( P, (num_products*num_markets, 1) ) ), 1 )
Z  = np.array( np.reshape( Z_10, (num_products, num_markets), order='F' ) )

# Log Normally Distributed Draws for Nu
nu = np.zeros( shape=( num_consumers * num_markets ) )
for i in range(num_consumers * num_markets):
    nu[i] = np.random.lognormal( 0.0, 1.0 )
nu = np.reshape( nu, (num_markets, num_consumers) )
```

```python
# Initial Guess for Mean Utility
guess_X = X
guess_Y = np.array( np.reshape( shares_10, (num_products*num_markets, 1) ) )

# Constructing Instruments
IV = f_ObtainInstrumentVariables( num_markets, num_products, guess_X )

# Commence Optimization of GMM Objective Function
estimated_market_shares = np.ones( (num_products, num_markets) )
estimated_mean_utility_iterated = np.ones( (num_products, num_markets) )

estimated_theta1 = f_ObtainOLSParameters( guess_X, guess_Y )
#estimated_theta1 = [1, 1, 1, 1]

# Using FMinSearch()
estimated_theta2 = sp.optimize.fmin( f_GMM, 1 )

# Using BFGS()
#estimated_theta2 = sp.optimize.minimize( f_GMM, 1, method='BFGS' )

# Using Basin-Hopping
#estimated_theta2 = sp.optimize.basinhopping( f_GMM, 1 )

# Calculating Estimated Prices and Market Shares
price_product1_10 = P[0,:]
price_product2_10 = P[1,:]
price_product3_10 = P[2,:]
price_product1_10 = np.reshape( price_product1_10, (-1, 1) )
price_product2_10 = np.reshape( price_product2_10, (-1, 1) )
price_product3_10 = np.reshape( price_product3_10, (-1, 1) )
share_product1_10 = estimated_market_shares[0,:]
share_product2_10 = estimated_market_shares[1,:]
share_product3_10 = estimated_market_shares[2,:]
share_product1_10 = np.reshape( share_product1_10, (-1, 1) )
share_product2_10 = np.reshape( share_product2_10, (-1, 1) )
share_product3_10 = np.reshape( share_product3_10, (-1, 1) )

# Variance-Covariance Matrix Determination
varcovar = np.diag( f_ObtainVarCov( estimated_theta2 ) )
stderror = np.sqrt( varcovar )

print('')
print('Estimation of Parameters and respective Standard Errors:')
print( 'Beta[0]:      ' + str( estimated_theta1[0] ) + '; (' + str( stderror[0] ) + ')
print( 'Beta[1]:      ' + str( estimated_theta1[1] ) + '; (' + str( stderror[1] ) + ')
print( 'Beta[2]:      ' + str( estimated_theta1[2] ) + '; (' + str( stderror[2] ) + ')
print( 'Alpha:        ' + str( estimated_theta1[3] ) + '; (' + str( stderror[3] ) + ')
print( 'Sigma Alpha: ' + str( estimated_theta2 ) + '; (' + str( stderror[4] ) + ') ' +
print('')
```

```
# Price Elasticities of Demand
##############################

# Generating Consumer Surplus
Xi = np.reshape( estimated_mean_utility_iterated , (num_markets*num_products , 1) ) - np.
f_ObtainConsumerSurplus( x1_10 , P_opt_10 , Xi , alphas_10 , estimated_theta1[0:3] , True )

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_3 = np.zeros( shape=( num_markets * num_products , 1 ) )
for i in range(num_markets * num_products):
    mc_3[i] = gamma[0] * 1 + gamma[1] * w_3[i] + gamma[2] * Z_3[i] + eta_3[i]

# Generating Profits
profit_product1_10 = np.zeros( shape=( num_markets , 1 ) )
profit_product2_10 = np.zeros( shape=( num_markets , 1 ) )
profit_product3_10 = np.zeros( shape=( num_markets , 1 ) )
for i in range( num_markets ):
    profit_product1_10[i] = ( price_product1_10[i] - mc_3[i*num_products] ) * num_consu
    profit_product2_10[i] = ( price_product2_10[i] - mc_3[i*num_products+1] ) * num_co
    profit_product3_10[i] = ( price_product3_10[i] - mc_3[i*num_products+2] ) * num_co

# Generate Histogram
print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 10
fig , histogram = plt.subplots(1, 3, sharey=False , tight_layout=True)
N, bins , patches = histogram[0].hist(profit_product1_3 , bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac , ipatch in zip( fracs , patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins , patches = histogram[1].hist(profit_product2_3 , bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac , ipatch in zip( fracs , patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins , patches = histogram[2].hist(profit_product3_3 , bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac , ipatch in zip( fracs , patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')


print( 'Section 3' )
print( 'Question 1' )
print('')

# Dataset Characteristics
num_markets   = 100
num_products  = 3
num_consumers = 500

# Import Dataset
dataset_file = '100markets3products.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

print( sp.io.whosmat( dataset_file ) )
print('')

# Dataset Characteristics
print( 'Markets:   ' + str( num_markets ) )
print( 'Products:  ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Generating New ndarray Variables from Dataset
x1_3     = np.array( dataset_raw['x1'] )
xi_all_3 = np.array( dataset_raw['xi_all'] )
w_3      = np.array( dataset_raw['w'] )
eta_3    = np.array( dataset_raw['eta'] )
Z_3      = np.array( dataset_raw['Z'] )
alphas_3 = np.array( dataset_raw['alphas'] )
P_opt_3  = np.array( dataset_raw['P_opt'] )
shares_3 = np.array( dataset_raw['shares'] )

print( 'x1     ' + str(x1_3.shape) )
print( 'xi_all ' + str(xi_all_3.shape) )
print( 'w      ' + str(w_3.shape) )
print( 'eta    ' + str(eta_3.shape) )
print( 'Z      ' + str(Z_3.shape) )
print( 'alphas ' + str(alphas_3.shape) )
print( 'P_opt  ' + str(P_opt_3.shape) )
print( 'shares ' + str(shares_3.shape) )
print('')

# Set Seed Value for Future Random Draws
np.random.seed( 1337 )
```

```
# Initial Setup for Guess of Mean Utility
P  = np.array( P_opt_3 )
X  = np.array( x1_3 )
X  = np.concatenate( ( X, np.reshape( P, (num_products*num_markets, 1) ) ), 1 )
Z  = np.array( np.reshape( Z_3, (num_products, num_markets), order='F' ) )

# Log Normally Distributed Draws for Nu
nu = np.zeros( shape=( num_consumers * num_markets ) )
for i in range(num_consumers * num_markets):
    nu[i] = np.random.lognormal( 0.0, 1.0 )
nu = np.reshape( nu, (num_markets, num_consumers) )

# Initial Guess for Mean Utility
guess_X = X
guess_Y = np.array( np.reshape( shares_3, (num_products*num_markets, 1) ) )

# Constructing Instruments
def f_ObtainInstrumentVariablesW( X ):
    IV = np.ones( (num_markets*num_products, 5) )
    IV[:, 0:3] = X[:, 0:3]
    X_23 = np.reshape( X[:, 1:3], (num_markets, num_products, 2) )
    Z_45 = np.ones( (num_markets, num_products, 2 ) )
    for i in range(2):
        for j in range(num_products):
            Z_45[:, j, i] = np.mean( np.delete( X_23[:, :, i], j, axis=1 ), axis=1 )
        IV[:, 3 + i] = np.reshape( Z_45[:, :, i], (num_markets*num_products, 1) )[:, 0]
    IV = np.concatenate( (IV, w_3), axis=1 )
    return IV

IV = np.zeros( (num_markets*num_products, 6) )
IV = f_ObtainInstrumentVariablesW( guess_X )

# Commence Optimization of GMM Objective Function
estimated_market_shares = np.ones( (num_products, num_markets) )
estimated_mean_utility_iterated = np.ones( (num_products, num_markets) )

estimated_theta1 = f_ObtainOLSParameters( guess_X, guess_Y )
#estimated_theta1 = [1, 1, 1, 1]

# Using FMinSearch()
estimated_theta2 = sp.optimize.fmin( f_GMM, 1 )

# Using BFGS()
#estimated_theta2 = sp.optimize.minimize( f_GMM, 1, method='BFGS' )

# Calculating Estimated Prices and Market Shares
price_product1_3 = P[0,:]
price_product2_3 = P[1,:]
price_product3_3 = P[2,:]
price_product1_3 = np.reshape( price_product1_3, (-1, 1) )
```

```
price_product2_3 = np.reshape( price_product2_3, (-1, 1) )
price_product3_3 = np.reshape( price_product3_3, (-1, 1) )
share_product1_3 = estimated_market_shares[0,:]
share_product2_3 = estimated_market_shares[1,:]
share_product3_3 = estimated_market_shares[2,:]
share_product1_3 = np.reshape( share_product1_3, (-1, 1) )
share_product2_3 = np.reshape( share_product2_3, (-1, 1) )
share_product3_3 = np.reshape( share_product3_3, (-1, 1) )

# Variance-Covariance Matrix Determination
varcovar = np.diag( f_ObtainVarCov( estimated_theta2 ) )
stderror = np.sqrt( varcovar )

print('')
print('Estimation of Parameters and respective Standard Errors:')
print( 'Beta[0]:      ' + str( estimated_theta1[0] ) + '; (' + str( stderror[0] ) + ') '
print( 'Beta[1]:      ' + str( estimated_theta1[1] ) + '; (' + str( stderror[1] ) + ') '
print( 'Beta[2]:      ' + str( estimated_theta1[2] ) + '; (' + str( stderror[2] ) + ') '
print( 'Alpha:        ' + str( estimated_theta1[3] ) + '; (' + str( stderror[3] ) + ') '
print( 'Sigma Alpha: ' + str( estimated_theta2 ) + '; (' + str( stderror[4] ) + ') ' +
print('')

# Price Elasticities of Demand
##############################

# Generating Consumer Surplus
Xi = np.reshape( estimated_mean_utility_iterated, (num_markets*num_products, 1) ) - np.
f_ObtainConsumerSurplus( x1_3, P_opt_3, Xi, alphas_3, estimated_theta1[0:3], True )

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_3 = np.zeros( shape=( num_markets * num_products, 1 ) )
for i in range(num_markets * num_products):
    mc_3[i] = gamma[0] * 1 + gamma[1] * w_3[i] + gamma[2] * Z_3[i] + eta_3[i]

# Generating Profits
profit_product1_3 = np.zeros( shape=( num_markets, 1 ) )
profit_product2_3 = np.zeros( shape=( num_markets, 1 ) )
profit_product3_3 = np.zeros( shape=( num_markets, 1 ) )
for i in range( num_markets ):
    profit_product1_3[i] = ( price_product1_3[i] - mc_3[i*num_products] ) * num_consum
    profit_product2_3[i] = ( price_product2_3[i] - mc_3[i*num_products+1] ) * num_consu
    profit_product3_3[i] = ( price_product3_3[i] - mc_3[i*num_products+2] ) * num_consu

# Generate Histogram
print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 25
fig, histogram = plt.subplots(1, 3, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(profit_product1_3, bins=num_bins)
fracs = N / N.max()
```

```
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(profit_product2_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(profit_product3_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')

#num_markets = 10
##############################
# Dataset Characteristics
num_markets = 10

# Import Dataset
dataset_file = '10markets3products.mat'
dataset_raw  = sp.io.loadmat( dataset_file )

print( sp.io.whosmat( dataset_file ) )
print('')

print( 'Markets:   ' + str( num_markets ) )
print( 'Products:  ' + str( num_products ) )
print( 'Consumers: ' + str( num_consumers ) )
print('')

# Generating New ndarray Variables from Dataset
x1_10     = np.array( dataset_raw['x1'] )
xi_all_10 = np.array( dataset_raw['xi_all'] )
w_10      = np.array( dataset_raw['w'] )
eta_10    = np.array( dataset_raw['eta'] )
Z_10      = np.array( dataset_raw['Z'] )
alphas_10 = np.array( dataset_raw['alphas'] )
P_opt_10  = np.array( dataset_raw['P_opt'] )
```

```
shares_10 = np.array( dataset_raw['shares'] )

print( 'x1      ' + str(x1_10.shape) )
print( 'xi_all  ' + str(xi_all_10.shape) )
print( 'w       ' + str(w_10.shape) )
print( 'eta     ' + str(eta_10.shape) )
print( 'Z       ' + str(Z_10.shape) )
print( 'alphas  ' + str(alphas_10.shape) )
print( 'P_opt   ' + str(P_opt_10.shape) )
print( 'shares  ' + str(shares_10.shape) )
print('')

# Set Seed Value for Future Random Draws
np.random.seed( 1337 )

# Initial Setup for Guess of Mean Utility
P  = np.array( P_opt_10 )
X  = np.array( x1_10 )
X  = np.concatenate( ( X, np.reshape( P, (num_products*num_markets, 1) ) ), 1 )
Z  = np.array( np.reshape( Z_10, (num_products, num_markets), order='F' ) )

# Log Normally Distributed Draws for Nu
nu = np.zeros( shape=( num_consumers * num_markets ) )
for i in range(num_consumers * num_markets):
    nu[i] = np.random.lognormal( 0.0, 1.0 )
nu = np.reshape( nu, (num_markets, num_consumers) )

# Initial Guess for Mean Utility
guess_X = X
guess_Y = np.array( np.reshape( shares_10, (num_products*num_markets, 1) ) )

# Constructing Instruments
def f_ObtainInstrumentVariablesW10( X ):
    IV = np.ones( (num_markets*num_products, 5) )
    IV[:, 0:3] = X[:, 0:3]
    X_23 = np.reshape( X[:, 1:3], (num_markets, num_products, 2) )
    Z_45 = np.ones( (num_markets, num_products, 2 ) )
    for i in range(2):
        for j in range(num_products):
            Z_45[:, j, i] = np.mean( np.delete( X_23[:, :, i], j, axis=1 ), axis=1 )
        IV[:, 3 + i] = np.reshape( Z_45[:, :, i], (num_markets*num_products, 1) )[:, 0]
    IV = np.concatenate( (IV, w_10), axis=1 )
    return IV

IV = f_ObtainInstrumentVariablesW10( guess_X )

# Commence Optimization of GMM Objective Function
estimated_market_shares = np.ones( (num_products, num_markets) )
estimated_mean_utility_iterated = np.ones( (num_products, num_markets) )
```

```
estimated_theta1 = f_ObtainOLSParameters( guess_X, guess_Y )
#estimated_theta1 = [1, 1, 1, 1]

# Using FMinSearch()
estimated_theta2 = sp.optimize.fmin( f_GMM, 1 )

# Using BFGS()
#estimated_theta2 = sp.optimize.minimize( f_GMM, 1, method='BFGS' )

# Using Basin-Hopping
#estimated_theta2 = sp.optimize.basinhopping( f_GMM, 1 )

# Calculating Estimated Prices and Market Shares
price_product1_10 = P[0,:]
price_product2_10 = P[1,:]
price_product3_10 = P[2,:]
price_product1_10 = np.reshape( price_product1_10, (-1, 1) )
price_product2_10 = np.reshape( price_product2_10, (-1, 1) )
price_product3_10 = np.reshape( price_product3_10, (-1, 1) )
share_product1_10 = estimated_market_shares[0,:]
share_product2_10 = estimated_market_shares[1,:]
share_product3_10 = estimated_market_shares[2,:]
share_product1_10 = np.reshape( share_product1_10, (-1, 1) )
share_product2_10 = np.reshape( share_product2_10, (-1, 1) )
share_product3_10 = np.reshape( share_product3_10, (-1, 1) )

# Variance-Covariance Matrix Determination
varcovar = np.diag( f_ObtainVarCov( estimated_theta2 ) )
stderror = np.sqrt( varcovar )

print('')
print('Estimation of Parameters and respective Standard Errors:')
print( 'Beta[0]:     ' + str( estimated_theta1[0] ) + '; (' + str( stderror[0] ) + ')
print( 'Beta[1]:     ' + str( estimated_theta1[1] ) + '; (' + str( stderror[1] ) + ')
print( 'Beta[2]:     ' + str( estimated_theta1[2] ) + '; (' + str( stderror[2] ) + ')
print( 'Alpha:       ' + str( estimated_theta1[3] ) + '; (' + str( stderror[3] ) + ')
print( 'Sigma Alpha: ' + str( estimated_theta2 ) + '; (' + str( stderror[4] ) + ') ' +
print('')

# Price Elasticities of Demand
##############################

# Generating Consumer Surplus
Xi = np.reshape( estimated_mean_utility_iterated, (num_markets*num_products, 1) ) - np.
f_ObtainConsumerSurplus( x1_10, P_opt_10, Xi, alphas_10, estimated_theta1[0:3], True )

# Generating Marginal Costs
gamma   = np.array( [2, 1, 1] )
mc_3 = np.zeros( shape=( num_markets * num_products, 1 ) )
for i in range(num_markets * num_products):
    mc_3[i] = gamma[0] * 1 + gamma[1] * w_3[i] + gamma[2] * Z_3[i] + eta_3[i]

# Generating Profits
profit_product1_10 = np.zeros( shape=( num_markets, 1 ) )
profit_product2_10 = np.zeros( shape=( num_markets, 1 ) )
profit_product3_10 = np.zeros( shape=( num_markets, 1 ) )
for i in range( num_markets ):
    profit_product1_10[i] = ( price_product1_10[i] - mc_3[i*num_products] ) * num_consu
    profit_product2_10[i] = ( price_product2_10[i] - mc_3[i*num_products+1] ) * num_co
    profit_product3_10[i] = ( price_product3_10[i] - mc_3[i*num_products+2] ) * num_co

# Generate Histogram
print( 'Generating Histogram of Distribution for Profits ...' )
num_bins = 10
fig, histogram = plt.subplots(1, 3, sharey=False, tight_layout=True)
N, bins, patches = histogram[0].hist(profit_product1_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[1].hist(profit_product2_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
N, bins, patches = histogram[2].hist(profit_product3_3, bins=num_bins)
fracs = N / N.max()
norm  = colors.Normalize( fracs.min(), fracs.max() )
for ifrac, ipatch in zip( fracs, patches ):
    color = plt.cm.viridis( norm( ifrac ) )
    ipatch.set_facecolor( color )
plt.xlabel('Profits');
plt.ylabel('Frequency');
plt.show()
print('')

# EOF
```