ECON 7510 - Problem Set #1

Julien Manuel Neves

October 27, 2018

Part 1

From the problem set description, we start by assuming that each consumer i has utility for product j as a function of price and product quality, δ_j of the following form

$$u_{ij} = \delta_j - \alpha_i p_j$$

where the distribution of consumer price sensitivity, α_i, is exponential with parameter $\lambda = 4 \cdot 10^{-6}$. We further assume that $\delta_j = x_j\beta + \xi_j$ and that x_j is uncorrelated with ξ_j.

In this model, i choose the j that provides the highest utility. This implies that if $p_j > p_k$, we need $\delta_j > \delta_k$ otherwise no one would ever pick j. In the same vein, it is straightforward to see that the share of good j will depend only on its close neighbors in the price dimension. Hence, before deriving the result, we sort the data from lowest to highest price.

As mentionned above, to determine j share we only need to compare its u_{ij} to $j-1$ and $j+1$. In fact, i will choose j if and only if

$$u_{i,j} > u_{i,j-1} \text{ and } u_{i,j} > u_{i,j+1}$$

This boils down to the following condition for α_i

$$\frac{\delta_{j+1} - \delta_j}{p_{j+1} - p_j} < \alpha_i < \frac{\delta_j - \delta_{j-1}}{p_j - p_{j-1}}$$

and therefore the share of good j is given by

$$\begin{aligned} s_j &= \int_{\frac{\delta_{j+1}-\delta_j}{p_{j+1}-p_j}}^{\frac{\delta_j-\delta_{j-1}}{p_j-p_{j-1}}} f(x)dx \\ &= F\left(\frac{\delta_j-\delta_{j-1}}{p_j-p_{j-1}}\right) - F\left(\frac{\delta_{j+1}-\delta_j}{p_{j+1}-p_j}\right) \end{aligned}$$

where $F(x) = 1 - e^{-\lambda x}$.

Since we know s_j and p_j, we can recursively solve for δ_j, by noting that the following relationship holds

$$\ln\left(\sum_{k=0}^{j} s_k\right) = -\lambda\left(\frac{\delta_{j+1}-\delta_j}{p_{j+1}-p_j}\right)$$

Using our δ_j, we can compute our estimate for β with simple OLS since the problem assumes that $\mathbb{E}\{x_j\xi_j\} = 0$. The results are stated in Table 1.

Table 1: Vertical model - Demand side only

	Demand side
(Intercept)	1.4370×10^8
	$[1.0166 \times 10^8 \ 1.8575 \times 10^8]$
Horsepower / Weight	1.6125×10^9
	$[7.9606 \times 10^8 \ 2.4291 \times 10^9]$
AC	5.7044×10^7
	$[4.3411 \times 10^7 \ 7.0676 \times 10^7]$
Firms FE	Yes
N	131
R^2	0.7195

[ ] 95% confidence interval

Note that while the regression in Table 1 includes fixed effects for the firm, I simply don't report their values for the sake of conciseness. Moreover, it is clear from our estimate that the coefficients values are immense which makes them difficult to interpret.

Part 2

For the model, if two goods, for example j and k, have the same price and strictly positive shares then $\delta_k = \delta_j$. As such, we have $u_{ik} = u_{ij}$ for all consumer for good j and k. If it were not the case, one good would have zero market share. This is a fairly extreme assumption.

Looking at Table 2, we can see that the model insinuates rather odd own and cross price elasticities.

Table 2: Vertical model - Price elasticities

Car\Car	15	16	17	18	19	20	21	22	23	24
15	-0.0948	0.0003	0	0	0	0	0	0	0	0
16	0.0003	-0.0003	0.0000	0	0	0	0	0	0	0
17	0	0.0001	-0.0671	0.0673	0	0	0	0	0	0
18	0	0	0.0050	-0.0154	0.0104	0	0	0	0	0
19	0	0	0	0.0711	-0.5703	0.4997	0	0	0	0
20	0	0	0	0	0.1349	-0.1477	0.0126	0	0	0
21	0	0	0	0	0	0.0235	-0.0438	0.0202	0	0
22	0	0	0	0	0	0	1.2703	-1.8608	0.5881	0
23	0	0	0	0	0	0	0	0.0175	-0.0180	0.0004
24	0	0	0	0	0	0	0	0	0.0001	-0.0390

This weird substitution pattern stems from the fact that cross price elasticities for any given good is going to be equal to zero apart from its direct price neighbors. It is straightforward to show this fact by noting that $s_j = F\left(\frac{\delta_j - \delta_{j-1}}{p_j - p_{j-1}}\right) - F\left(\frac{\delta_{j+1} - \delta_j}{p_{j+1} - p_j}\right)$ is not a function of any price apart from p_j, p_{j-1}, and p_{j+1}.

Moreover, the own-price elasticities are pretty close to each other accross the price spectrum. This is not what we would expect in reality (people buying high-end cars should be less price sensitive than people buying low-end cars).

Part 3

We know turn our attention to the supply side of our model. We are given the following equation for marginal cost $mc_j = x_j\gamma + \eta q_j + \omega_j$. Sadly, we don't have the actual value for the marginal cost. We can circumvent this issue by relating marginal cost to price by assuming some pricing strategy for the firms. As a matter of fact, these pricing strategy will result in a markup over marginal cost given by the following formulas:

- Marginal cost pricing:

$$p_j = mc_j$$

- Single product firms:

$$p_j = mc_j + \Delta^{-1}s$$

where $\Delta_{jk} = \begin{cases} \frac{\partial s_j}{\partial p_k} & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$ and s is the vector of shares.

- Multiproduct firms:

$$p_j = mc_j + \Delta^{-1} s$$

  where $\Delta_{jk} = \begin{cases} \frac{\partial s_j}{\partial p_k} & \text{if } k \in \text{firm}(j) \\ 0 & \text{otherwise} \end{cases}$ and s is the vector of shares.

- Perfect collusion:

$$p_j = mc_j + \Delta^{-1} s$$

  where $\Delta_{jk} = \frac{\partial s_j}{\partial p_k}$ and s is the vector of shares.

From the previous question, it is easy to to compute Δ and therefore mc_j.

With this in mind, we can revert back to $mc_j = x_j\gamma + \eta q_j + \omega_j$ and estimate it like we would for $\delta_j = x_j\beta + \xi_j$. The only issue we are facing now is the potential endogeneity of q_j.

To remedy the situation, we need to find an instrument for q_j. There is plenty potential instruments, but after playing around for a while, I decide to follow BLP and use $\sum_{k \in \text{firm}(j)}(x_k - x_j)$ and $\sum_{k \notin \text{firm}(j)} x_k$. To avoid any problem with collinearity and matrix inversion, I use the function `licols()` to reduce our set of instruments $Z = [x_j, \sum_{k \in \text{firm}(j)}(x_k - x_j), \sum_{k \notin \text{firm}(j)} x_k]$ to a full column rank matrix. Note that the function `create_iv()` includes the method from Gandhi and Houde (2018).

We can now define our moments equations:

$$\mathbb{E}\left\{ Z \begin{bmatrix} \xi_j \\ \omega_j \end{bmatrix} \right\} = 0$$

Using these moments conditions, we can run a GMM estimation using the different pricing strategy by stacking ξ_j and ω_j. Results are given in Table 3.

Table 3: Vertical model - supply side

	Marginal cost	**Single product firms**	**Multiproduct firms**	**Collusion**
Demand				
(Intercept)	1.4370×10^8	1.4370×10^8	1.4370×10^8	1.4370×10^8
	$[1.0166 \times 10^8\ 1.8575 \times 10^8]$	$[1.0166 \times 10^8\ 1.8575 \times 10^8]$	$[1.0166 \times 10^8\ 1.8575 \times 10^8]$	$[1.0166 \times 10^8\ 1.8575 \times 10^8]$
Horsepower / Weight	1.6125×10^9	1.6125×10^9	1.6125×10^9	1.6125×10^9
	$[7.9606 \times 10^8\ 2.4291 \times 10^9]$	$[7.9606 \times 10^8\ 2.4291 \times 10^9]$	$[7.9606 \times 10^8\ 2.4291 \times 10^9]$	$[7.9606 \times 10^8\ 2.4291 \times 10^9]$
AC	5.7044×10^7	5.7044×10^7	5.7044×10^7	5.7044×10^7
	$[4.3411 \times 10^7\ 7.0676 \times 10^7]$	$[4.3411 \times 10^7\ 7.0676 \times 10^7]$	$[4.3411 \times 10^7\ 7.0676 \times 10^7]$	$[4.3411 \times 10^7\ 7.0676 \times 10^7]$
Firms FE	Yes	Yes	Yes	Yes
N	131	131	131	131
R^2	0.7195	0.7195	0.7195	0.7195
Supply (Intercept)	-1.4439×10^4	2.1439×10^8	2.4358×10^8	3.3766×10^{11}
	$[-4.1412 \times 10^7\ 1.2534 \times 10^4]$	$[-3.2494 \times 10^8\ 7.5373 \times 10^8]$	$[-4.4677 \times 10^8\ 9.3394 \times 10^8]$	$[2.8871 \times 10^{11}\ 3.8660 \times 10^{11}]$
Horsepower/Weight	4.9504×10^5	6.9673×10^8	1.0852×10^9	2.3462×10^{12}
	$[2.2007 \times 10^5\ 7.7001 \times 10^5]$	$[-4.8014 \times 10^9\ 6.1948 \times 10^9]$	$[-5.9525 \times 10^9\ 8.1229 \times 10^9]$	$[1.8472 \times 10^{12}\ 2.8451 \times 10^{12}]$
AC	1.0480×10^4	-1.0186×10^8	-1.2394×10^8	6.2855×10^{10}
	$[4.1786 \times 10^3\ 1.6781 \times 10^4]$	$[-2.2786 \times 10^8\ 2.4142 \times 10^7]$	$[-2.8523 \times 10^8\ 3.7338 \times 10^7]$	$[5.1420 \times 10^{10}\ 7.4290 \times 10^{10}]$
Quantity	0.0301	-2070.9	-2516.3	1.0040×10^6
	$[-0.1135\ 0.1736]$	$[-4.9414 \times 10^3\ 799.5432]$	$[-6190.5\ 1158.0]$	$[7.4347 \times 10^5\ 1.2645 \times 10^6]$
Firm FE	Yes	Yes	Yes	Yes
N	131	131	131	131
R^2	0.8032	0.3507	0.3821	0.8279

[ ] 95% confidence interval

Note that the coefficients are all over the place, and it is not worth it to interpret them. Seems like our data set does not contain enough information to identify most of the parameters.

Part 4

Potentially what we would want to identify the different pricing strategy would be some sort of shock on the supply side. For example, the exogeneous entry or exit of some product. Then, we could test which pricing equation fares better.

Short of this, we could simply run some goodness of fit test to help us select the best approximation. For example, we could start with the null hypothesis that firm use marginal cost pricing and compute the likelihood ratio statistic of the other pricing decision. Rejection of the null would imply that the firms have potentially some market power that can be explained by some Nash-Bertrand equilibrium.

Part 5

We now assume that each consumer i has utility for product j as a function of price and product quality, δ_j of:

$$u_{ij} = \delta_j + \epsilon_{ij}$$

where $\delta_j = x_j\beta - \alpha p_j + \xi_j$, ξ_j is the unosbervable characteristics of good j and ϵ_{ij} follows a type I extreme value distribution.

As shown in class, we have that the share of good j is given by

$$s_j = \frac{e^{\delta_j}}{1 + \sum_k e^{\delta_k}}$$

or, equivalently,

$$\ln(s_j) - \ln(s_0) = x_j\beta - \alpha p_j + \xi_j$$

Armed with this regression, we can now estimate $[\beta, \alpha]$. Like with the supply side, we now have some endogeneity problem. To solve this, we use the same set of instruments, $Z = [x_j, \sum_k |x_j - x_k|, \sum_k (x_j - x_k)^2]$, and run an IV regression to get back both $[\beta, \alpha]$ and ξ. The result from our IV regression are reported in Table 4.

Moreover, Table 4 includes the estimation of demand and supply parameters assuming Nash-Bertrand equilibrium with multiproduct firms. Recall that with multiproduct firms, Nash-Bertrand equilibrium yields

$$p = mc + \Delta^{-1}s = \Delta^{-1}s + x\gamma + \eta q + \omega$$

where $\Delta_{jk} = \begin{cases} \frac{\partial s_j}{\partial p_k} & \text{if } k \in \text{firm}(j) \\ 0 & \text{otherwise} \end{cases}$ and s is the vector of shares. We can solve explicitly for Δ by noting that $\frac{\partial s_j}{\partial p_k} = \begin{cases} -\alpha s_j(1 - s_j) & \text{if } j = k \\ \alpha s_j s_k & \text{otherwise} \end{cases}$. Let $\Delta^{-1}s = b(\alpha)$ since, unlike the vertical model, it depends on α in a non-linear fashion.

We can now define our moments equations:

$$\mathbb{E}\left\{ \begin{matrix} Z\xi_j \\ Z\omega_j \end{matrix} \right\} = 0$$

or, equivalently

$$\mathbb{E}\left\{ \begin{matrix} Z(\delta_j - x_j\beta + \alpha p_j) \\ Z(p_j - b_j(\alpha) - x_j\gamma - \eta q_j) \end{matrix} \right\} = 0$$

where Z is the set of instruments. Using these statcked moments conditions, we can use our usual GMM technique to estimate $\theta = [\alpha, \beta, \gamma, \eta]$. Since α enters non-linearly, to solve for it I use `fmincon()` to minize the objective function with respect to α and then solve for the rest of parameters using least squares techniques.

The only issue that I'm facing now is how to compute the standard errors of our estimate. Since, α enters non-linearly, I need to find the gradient of s_j with θ to be able to compute the GMM covariance matrix. Estimation results are reported in Table 4.

Table 4: Vertical model - supply side

	Demand only	**Multiple product firms**
(Intercept)	-6.6124	-6.7250
	$[-8.0979\ -5.1270]$	$[-8.0552\ -5.3949]$
Horesepower/Weight	-8.5267	-2.9597
	$[-43.7682\ 26.7149]$	$[-28.7903\ 22.8710]$
AC	-0.3297	-0.2147
	$[-0.9764\ 0.3169]$	$[-0.6459\ 0.2166]$
Price (α)	2.7842×10^{-5}	4.0202×10^{-5}
	$[-2.1040 \times 10^{-5}\ 7.6723 \times 10^{-5}]$	$[3.8732 \times 10^{-5}\ 4.1672 \times 10^{-5}]$
FE	Yes	Yes
N	131	131
R^2	0.5696	0.5134
(Intercept)		1.0631×10^4
		$[-1.6342 \times 10^4\ 3.7604 \times 10^4]$
Horesepower/Weight		4.9504×10^5
		$[2.2007 \times 10^5\ 7.7001 \times 10^5]$
AC		1.0480×10^4
		$[4.1786 \times 10^3\ 1.6781 \times 10^4]$
Quantity		0.0301
		$[-0.1135\ 0.1736]$
FE		Yes
N	131	131
R^2		0.8017

[ ] 95% confidence interval

Again, we report the elasticities for a subset of the product in Table 5.

Table 5: Logit model - Price elasticities

Car\Car	15	16	17	18	19	20	21	22	23	24
15	-0.2057	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
16	5.9558×10^{-4}	-0.2103	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
17	5.9558×10^{-4}	6.2258×10^{-4}	-0.2420	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
18	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	-0.2430	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
19	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	-0.2445	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
20	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	-0.2447	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
21	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	-0.2461	2.5681×10^{-6}	8.6135×10^{-5}	3.2681×10^{-4}
22	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	-0.2476	8.6135×10^{-5}	3.2681×10^{-4}
23	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	-0.2496	3.2681×10^{-4}
24	5.9558×10^{-4}	6.2258×10^{-4}	4.1449×10^{-5}	5.5583×10^{-5}	8.1653×10^{-5}	3.0235×10^{-4}	1.6152×10^{-4}	2.5681×10^{-6}	8.6135×10^{-5}	-0.2629

Looking at these elasticities, we have an improvment compared to the previous vertical model where most cross price elasticities where equal to 0. But now, we the drawbacks of logit model, i.e. IIA property. This implies that odds of choosing some good j over k does not depend on other alternatives which is at odd with what we expect.

Notice that the own price elasticity of good j depends solely on the price and share of j, while the cross price elasticities of j depends on the price and share of k. Therefore, two goods with the same shares will have the same cross price elasticities for any other good and the same markup. Moreover, it is clear to see that from the cross price elasticities in Table 5 that an increase in price for some good k will push consumer towards the most bought good. This is fairly odd as we don't expect people buying expensive car to switch to cheap cars if the price of high end cars rise slightly.

Part 6

All three proposed model could improve our substitution patterns. First, the Pure Characteristics model assume away the logit error term ϵ_{ij}. Since this is the source of the IIA problem with the logit model, it is straightforward to see that the Pure Characteristics model would improve our elasticities.

Moreover, the Nested Logit can be use to break the IIA by allowing correlation between some choices. For example, we could put high-end cars in the same nest to better capture how consumer choose between alternatives. In the same fashion, the Multinomial probit can also break IIA by allowing correlation between the error terms. In both cases, we improve on the substitution pattern.

Part 7

We now assume that each consumer i has utility for product j as a function of price and product quality, δ_j of:

$$u_{ij} = \delta_j - \alpha_i p_j + \epsilon_{ij}$$

where $\delta_j = x_j\beta + \xi_j$, ϵ_{ij} follows a type I extreme value distribution and $\alpha_i = \frac{1}{y_i}$ where y_i is income and distributed as a lognormal.

As shown in class, we have that the share of probability that i buys good j is given by

$$P_{ij} = \frac{e^{\delta_j - \alpha_i p_j}}{1 + \sum_k e^{\delta_k - \alpha_i p_k}}$$

To obtain the shares, we need to integrate over the consumers, i.e.

$$g_j = \int \frac{e^{\delta_j - \alpha_i p_j}}{1 + \sum_k e^{\delta_k - \alpha_i p_k}} d\Phi(\alpha_i)$$

where $\Phi(\cdot)$ is the lognormal distribution of income. In our context, it might be easy to integrate out the probabilities since we only have one α_i and that the lognormal is fully characterize by its two unknown parameters μ and σ. Usually, it is easier to simulate the α_i and take average of the probabilities. This is what we do in this problem set, by simulating ns y_i from a lognormal with μ and σ and solving for g_j with the following formula:

$$g_j = \frac{1}{ns} \sum_{i=1}^{ns} \frac{e^{\delta_j - \alpha_i p_j}}{1 + \sum_k e^{\delta_k - \alpha_i p_k}}$$

With the s_j in hands, we need to find the parameters to match them to the data. To do this, we first pick some μ and σ. Next, we solve for δ_j with a fixed point iteration

$$\delta^{l+1} = \delta^l + \ln(s) - \ln(g_j)$$

Recalling that $\delta_j = x_j\beta + \xi_j$, it is easy to see that we can solve for our unknowns by using the following moment condition $\mathbb{E}\{\xi_j \mid Z\} = 0$ or

$$\mathbb{E}\{Z\xi_j\} = 0$$

where Z is the set of instruments as defined previously. Note that ξ_j depends on β linearly since $\xi_j = \delta_j(\mu, \sigma) - x_j\beta$, but it depends on σ and μ we depend on them to find δ_j. Therefore, given μ and σ we can solve for β and ξ_j using least squares, but we can't use the same technique to solve for the non-linear parameters.

Therefore, what we do to solve for μ and σ is to minimize the following objective function

$$(Z'\xi(\mu, \theta))'W(Z'\xi(\mu, \sigma))$$

where W is some weighting matrix. In this problem, we let $W = Z'Z$ which is optimal if the ξ_j are homoskedastic.

To solve for this problem, we can simply use any optimization algorithm to minize the objective function.

Now, suppose that $\alpha_i = \alpha_1 + \frac{\alpha_2}{y_i}$. The first thing to note is that α_1 will enter δ_j linearly, and we can brush it aside as it is easy to deal with this. As for α_2, it simply an added non-linear parameter to deal with and we can take care of it like μ and σ.

The question that remains is, can we identify our parameters. Technically, if we have more instruments than parameters we should be in the clear. Past this requirement, we may face two problems. First, we might not have enough variation and power to identify both $[\beta, \alpha]$ and $[\alpha_2, \mu, \sigma]$. Second, since $Y_i = \frac{\alpha_i}{y_i}$ can be seen as a random variable, depending on the functional form of its distribution it might be simply impossible to seperatly identify all three parameters $[\alpha_2, \mu, \sigma]$.

Part 8

The estimation results for the model described in part 7 are reported in Table 6.

Table 6: BLP model - supply side

(Intercept)	-6.6124
	$[-8.0967\ -5.1281]$
Horesepower/Weight	-8.5261
	$[-43.7403\ 26.6881]$
AC	-0.3297
	$[-0.9758\ 0.3165]$
Price ($\alpha_!$)	2.7433×10^{-5}
	$[-2.1410 \times 10^{-5}\ 7.6276 \times 10^{-5}]$
Price (α_2)	0.0057
	$[-0.8140\ 0.8255]$
FE	Yes
N	131
R^2	0.5687

[ ] 95% confidence interval

It is important to note that for different starting value for our minimization algorithm, we find different value for α_2 and hence different $[\beta, \alpha_1]$. Figure 1 shows an histogram of the different values of α_2 obtained with a 100 starting point evenly spaced between 0 and 50.

Finally, we can look at the prices elasticities implied by the model as shown in Table 7.

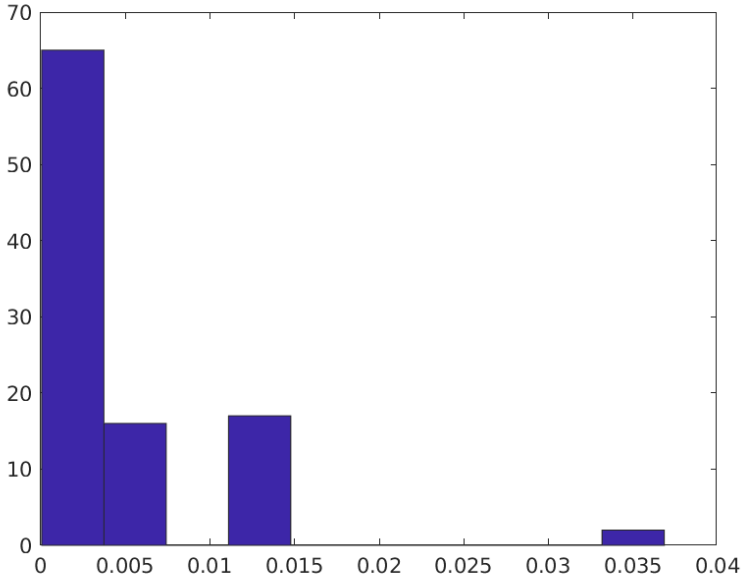

Figure 1: histogram of α_2 for different starting value

Table 7: BLP model - Price elasticities

Car\Price	15	16	17	18	19	20	21	22	23	24
15	-0.2006	6.1943×10^{-4}	4.1220×10^{-5}	5.5274×10^{-5}	8.1199×10^{-5}	3.0067×10^{-4}	1.6062×10^{-4}	2.5537×10^{-6}	8.5649×10^{-5}	3.2490×10^{-4}
16	5.9257×10^{-4}	-0.2098	4.1217×10^{-5}	5.5270×10^{-4}	8.1193×10^{-5}	3.0065×10^{-4}	1.6061×10^{-4}	2.5535×10^{-6}	8.5643×10^{-5}	3.2488×10^{-4}
17	5.9229×10^{-4}	6.1909×10^{-4}	-0.2413	5.5244×10^{-4}	8.1155×10^{-5}	3.0051×10^{-4}	1.6053×10^{-4}	2.5523×10^{-6}	8.5603×10^{-5}	3.2472×10^{-4}
18	5.9228×10^{-4}	6.1908×10^{-4}	4.1197×10^{-5}	-0.2423	8.1153×10^{-5}	3.0050×10^{-4}	1.6053×10^{-4}	2.5523×10^{-6}	8.5601×10^{-5}	3.2472×10^{-4}
19	5.9227×10^{-4}	6.1907×10^{-4}	4.1196×10^{-5}	5.5242×10^{-4}	-0.2438	3.0050×10^{-4}	1.6052×10^{-4}	2.5522×10^{-6}	8.5600×10^{-5}	3.2471×10^{-4}
20	5.9226×10^{-4}	6.1907×10^{-4}	4.1196×10^{-5}	5.5242×10^{-4}	8.1151×10^{-5}	-0.2440	1.6052×10^{-4}	2.5522×10^{-6}	8.5599×10^{-5}	3.2471×10^{-4}
21	5.9225×10^{-4}	6.1905×10^{-4}	4.1195×10^{-5}	5.5241×10^{-4}	8.1150×10^{-5}	3.0049×10^{-4}	-0.2454	2.5522×10^{-6}	8.5597×10^{-5}	3.2470×10^{-4}
22	5.9224×10^{-4}	6.1904×10^{-4}	4.1194×10^{-5}	5.5240×10^{-4}	8.1148×10^{-5}	3.0048×10^{-4}	1.6052×10^{-4}	-0.2469	8.5596×10^{-5}	3.2470×10^{-4}
23	5.9222×10^{-4}	6.1902×10^{-4}	4.1193×10^{-5}	5.5238×10^{-4}	8.1145×10^{-5}	3.0047×10^{-4}	1.6051×10^{-4}	2.5520×10^{-6}	-0.2489	3.2469×10^{-4}
24	5.9210×10^{-4}	6.1889×10^{-4}	4.1185×10^{-5}	5.5226×10^{-4}	8.1129×10^{-5}	3.0041×10^{-4}	1.6048×10^{-4}	2.5515×10^{-6}	8.5575×10^{-5}	-0.2621

It is clear now that we have slightly improved the pattern of substitution between goods. In fact, unlike the simple logit model, our new elasticities do not satisfy the IIA property which is what we desired. On the otherhand, the few data points we have and the lack of micro-moments does not help us.

Part 9

Recall that we assume that each consumer i has utility for product j as a function of price and product quality, δ_j of:

$$u_{ij} = x_j\beta + \xi_j - \alpha p_j + \epsilon_{ij}$$

where ϵ_{ij} follows a type I extreme value distribution and $\alpha = \alpha_1 + \frac{\alpha_2}{y_i}$ where y_i is income and distributed as a lognormal.

The micro-moments presented in the problem set can supplement our model by observing that the taste preference for some characteristics (Horesepower to weight) a function of individual specific characteristics. In our context, we account for this by modifying our β to include this possibility, i.e.

$$\beta_{i,hp/w} = \beta_{hp/w} + y_i \beta^0_{hp/w}$$

where $\beta^0_{hp/w}$ is the preference associated with observed consumer attributes.

Note that we could also include some unobservable idiosyncratic tastes in $\beta_{i,k}$.

Then, as previously described, our first moment is still the one that matches market shares. The new moment condition is given by

$$G^2(\theta) = \sum_j \frac{n_j}{ns} x_{j,hp/w} \left\{ \frac{1}{n_j} \sum_i y_i - \mathbb{E}\left\{ y \mid j, \theta \right\} \right\} - 0.7 \sigma_y \sigma_{x_{hp/w}} = 0$$

where n_j is the number of people that buy good j and $\mathbb{E}\left\{ y \mid j, \theta \right\}$ is the expected income of someone buying car j given θ and σ_y $\sigma_{x_{hp/w}}$ are variance of income and horesepower/weight. We can compute $\mathbb{E}\left\{ y \mid j, \theta \right\}$ using Bayes' rule and the moment derive in the first part of the model.

While useful for the random coefficient model, micro-moments have no impact in the logit model since it ignore unobserved consumer preference. Hence, it would not be useful to estimate the parameters or change the substitution patterns in part 5.

Code

Main file

```
%% Prepare Data
clc
clear

%% Change Folder
% Get folder where J0_prepare_data is located
folder = fileparts(which('prepare_data')) ;
% Change to parent folder and add subfolders to path
cd(folder);
cd('..');
addpath(genpath(cd));


%% Prepare data
```

```
prepare_data

load Dataset
params.number_product = size(Dataset.data,1);
params.mean_utility_tol = 1e-12;
params.max_ite = 5000;
params.M = 100000000;
params.nb_cars = size(Dataset.data.price,1);
params.IV_type = 'BLP';
params.hessian = 1;


% Sort by price
Dataset.data = sortrows(Dataset.data,1);

% Compute shares
Dataset.shares = Dataset.data.quantity/params.M;

%
dummy_firm = dummyvar(Dataset.data.firm);
prod_char = [Dataset.data.weight Dataset.data.hp Dataset.data.AC];
Dataset.X = [ones(params.nb_cars,1) Dataset.data.hp./Dataset.data.
    weight Dataset.data.AC dummy_firm(:,2:end)];
Dataset = create_iv(Dataset, params);

%% Vertical Model (Question 1)
params.lambda = 4e-6;
params.model = 'vertical';
params.specification = 'demand';

% Define
Dataset.Xd = Dataset.X;

% Estimate model
[est, Dataset] = vertical_model(Dataset, params);
result.vertical= est;

%% Question 2
% Elasticities
disp(result.vertical.demand.elasticities);


%% Question 3

Dataset.Xd = Dataset.X;
```

```
59 Dataset.Xs = [Dataset.X Dataset.data.quantity];
60
61
62 % Competition
63 params.specification = 'competition';
64 [est, Dataset] = vertical_model(Dataset, params);
65 result.vertical.competition = est;
66
67 % Single
68 params.specification = 'single';
69 [est, Dataset] = vertical_model(Dataset, params);
70 result.vertical.single = est;
71
72
73 % Multiple
74 params.specification = 'multiple';
75 [est, Dataset] = vertical_model(Dataset, params);
76 result.vertical.multiple = est;
77
78 % Collusion
79 params.specification = 'collusion';
80 [est, Dataset] = vertical_model(Dataset, params);
81 result.vertical.collusion = est;
82
83
84 %% Logit
85
86 params.model = 'logit';
87 params.specification = 'demand';
88
89 Dataset.Xd = [Dataset.X -Dataset.data.price];
90 [est, Dataset] = logit_model(Dataset, params);
91
92 result.logit= est;
93
94 % Multiple
95
96 params.nb_init = 20;
97 params.theta_start = 1;
98 params.lower_bound = 0;
99 params.upper_bound = 50;
100 params.nK = size(params.theta_start,1);
101
102 params.specification = 'multiple';
103 Dataset.Xd = [Dataset.X];
```

```
Dataset.Xs = [Dataset.X Dataset.data.quantity];
[est, Dataset] = logit_model(Dataset, params);
result.logit.multiple = est;

%% BLP
params.model = 'BLP';
params.specification = 'demand';
params.nb_draws = 1000;
params.income_mean = 35000;
params.income_sd = 45000;

params.nb_init = 100;
params.theta_start = 22;
params.lower_bound = 0;
params.upper_bound = 50;
params.nK = size(params.theta_start,1);

% Draw income from lognormal
mu = log((params.income_mean^2)/sqrt(params.income_sd^2+params.
    income_mean^2));
sigma = sqrt(log(params.income_sd^2/(params.income_mean^2)+1));

Draws.income = lognrnd(mu,sigma,1,params.nb_draws);

%
Dataset.Xd = [Dataset.X -Dataset.data.price];

[est, Dataset] = blp_model(Dataset, params, Draws);
result.blp = est;

for i = 1:params.nb_init
    B(i,:)=est{i}.alpha;
end
hist(B(:,2))
saveas(gcf, 'output/alpha2.png')

save output/result.mat
```

BLP_model

```
function [result, Dataset] = blp_model(Dataset, params, Draws)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: Estimation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%================ 1.1: Matrix of Theta ============================
%theta=[alpha; sigma_p; lambda]
nb_init = params.nb_init;
nK = params.nK;

% Set particular inital values
theta_start= params.theta_start;

% Set bounds
lower_bound = params.lower_bound;
upper_bound =params.upper_bound;

theta_init = linspace(lower_bound,upper_bound, nb_init);

% Combine inital value and random grid points
theta_init=[theta_start, theta_init];


%================ 1.2. Optimal GMM ============================
options = optimoptions('fmincon','Display','iter', ...
        'SpecifyObjectiveGradient',false,'CheckGradients',false,
            ...
        'FiniteDifferenceType', 'central', ...
        'FiniteDifferenceStepSize', 1e-10,'StepTolerance',1e-10,
            ...
        'MaxFunEval', inf, 'MaxIter', inf);

%%%%%% Find estimates using different initial values
theta_mat = zeros(nb_init, nK);

for i=1:params.nb_init
    % Estimation
     [theta_mat(i),~,~,~,~,~,hessian]  = fmincon('gmm_fun',
        theta_init(i), [], [], [], [], ...
                                lower_bound, upper_bound, [], options,
                                    ...
                                Dataset, params, Draws);

```

```
    params.hessian    = hessian;
    [fval, est] = gmm_fun(theta_mat(i), Dataset, params, Draws);
    result{i}= est;
    result{i}.start = theta_init(i);
    result{i}.theta = theta_mat(i);
    result{i}.fval = fval;
    clc;
end


end
```

create_iv

```
function Dataset = create_iv(Dataset, params)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
IV_type = params.IV_type;
firm = Dataset.data.firm;
prod_char = [Dataset.data.weight Dataset.data.hp Dataset.data.AC];
X = Dataset.X;

switch IV_type
    case 'BLP'
        for i = 1:params.nb_cars
            sum_within(i,:) = sum(prod_char(firm==firm(i),:),1);
            sum_outside(i,:) = sum(prod_char(firm~=firm(i),:),1);
        end
        sum_within = sum_within-prod_char;
        Dataset.IV = [X sum_within sum_outside];

        Dataset.IV =licols(Dataset.IV);
    case 'Houde'
        for i = 1:params.nb_cars
            sum_abs(i,:) = sum(abs(prod_char-prod_char(i,:)),1);
            sum_square(i,:) = sum((prod_char-prod_char(i,:)).^2,1)
                ;
        end
        Dataset.IV = [X sum_abs sum_square];

        Dataset.IV =licols(Dataset.IV);
    otherwise
        warning('No IV type specified');
end

```

```
Dataset.W = (Dataset.IV'*Dataset.IV)\eye(size(Dataset.IV,2));
Dataset.Pz =  Dataset.IV* Dataset.W * Dataset.IV';
end
```

get_markup

```
function [b, e, delta] = get_markup(Dataset, params, result, Draws
    )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

model = params.model;
specification = params.specification;

switch model
    case 'vertical'
        lambda = params.lambda;
        shares = Dataset.shares;
        price = Dataset.data.price;

        % Add outside option price and share
        shares = [1-sum(shares); shares];
        price = [0; price];

        % Compute mean utility for alpha ~ exponential
        share_cumsum = cumsum(shares);
        price_diff = diff(price);

        temp = share_cumsum(1:end-1,1)./(price_diff).^2;

        delta = zeros(size(temp,1)+1);

        delta(1:end-1,1:end-1) = -lambda*diag(temp) + delta(1:end
            -1,1:end-1);
        delta(2:end,2:end) = -lambda*diag(temp) + delta(2:end,2:
            end);
        delta(1:end-1,2:end) = lambda*diag(temp) + delta(1:end
            -1,2:end);
        delta(2:end,1:end-1) = lambda*diag(temp) + delta(2:end,1:
            end-1);


        delta = delta(2:end,2:end);
        price = price(2:end,1);
        shares = shares(2:end,1);
```

```
35          e = delta.*(price'./shares);
36
37      case 'logit'
38          shares = Dataset.shares;
39          price = Dataset.data.price;
40          alpha = result.demand.alpha;
41
42          delta = alpha*(shares.*shares'-diag(shares));
43          e = delta.*(price'./shares);
44      case 'BLP'
45          shares = Dataset.shares;
46          price = Dataset.data.price;
47          mean_utility = Dataset.mean_utility;
48          alpha = result.alpha;
49          yi = Draws.income;
50          nb_draws = size(yi,2);
51
52          Numerator = exp(mean_utility - alpha(2)*price./yi);
53          ind_prob = bsxfun(@rdivide, Numerator, 1+sum(Numerator,1))
                ;
54          delta = zeros(size(ind_prob,1));
55
56          for i = 1:nb_draws
57              temp = ind_prob(:,i);
58              D = (alpha(1)+alpha(2)/yi(:,i))*(temp.*temp'-diag(temp
                    ));
59
60              delta = delta + D;
61          end
62          delta = delta/nb_draws;
63          e = delta.*(price'./shares);
64  end
65
66  switch specification
67      case 'single'
68          b = (eye(size(delta)).*delta)\shares;
69      case 'multiple'
70          b  = (dummyvar(Dataset.data.firm)*dummyvar(Dataset.data.
                firm)'.*delta)\shares;
71      case 'collusion'
72          b = (ones(size(delta)).*delta)\shares;
73      otherwise
74          b = 0;
75  end
76
```

```

end
```

get_mean_utility

```
function mean_utility = get_mean_utility(Dataset, params, Draws,
    theta)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
model = params.model;
tol = params.mean_utility_tol;
max_ite = params.max_ite;

shares = Dataset.shares;
mean_utility = zeros(size(shares));

switch model
    case 'logit'
      mean_utility = log(shares)-log(1-sum(shares));
    case 'BLP'
        ite = 0;
        tol_crit = 1;
        while tol_crit>tol && ite <= max_ite
            shares_sim = get_shares(mean_utility, Dataset, params,
                theta, Draws);
            mean_utility = mean_utility + log(shares) - log(
                shares_sim);

            ite = ite + 1;
            tol_crit = norm(shares-shares_sim);
            %fprintf('Iteration: %1$d Criteria: %2$e \n', ite,
                tol_crit)
        end
    otherwise
        warning('No model')
end
end
```

get_shares

```
function market_share = get_shares(mean_utility, Dataset, params,
    theta, Draws)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
model = params.model;

```

```
switch model
    case 'logit'
        Numerator = exp(mean_utility);
        market_share = bsxfun(@rdivide, Numerator, 1+sum(Numerator
            ));
    case 'BLP'
        price = Dataset.data.price;
        yi = Draws.income;

        Numerator = exp(mean_utility - theta(1)*price./yi);
        ind_prob = bsxfun(@rdivide, Numerator, 1+sum(Numerator,1))
            ;
        market_share = mean(ind_prob,2) ;
    otherwise
        warning('No model')
end


end
```

gmm_fun

```
function [objective_val, est] = gmm_fun(theta, Dataset, params,
    Draws)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

IV = Dataset.IV;
Xd = Dataset.Pz*Dataset.Xd;
W = Dataset.W;
hessian = params.hessian;
tval = tinv(.95,size(Xd,1)-size(Xd,2));
se = sqrt(diag(inv(hessian)));
ci_theta = [theta-se*tval theta+se*tval];

mean_utility = get_mean_utility(Dataset, params, Draws, theta);

[beta,ci,xi,~,stats] = regress(mean_utility,Xd);
est.beta = [beta; theta];
est.stats = stats;
est.ci = [ci; ci_theta];
est.alpha = [beta(end) theta];
Dataset.mean_utility;

[b,e,~] = get_markup(Dataset, params, est, Draws);
```

```
est.elasticities = e;


objective_val = (IV' * xi)'* W * (IV' * xi);


end
```

logit_model

```
function [result, Dataset] = logit_model(Dataset, params)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
specification = params.specification;
mean_utility = get_mean_utility(Dataset, params);


switch specification
    case 'demand'
        % OLS of mean_utility
        Xd = Dataset.Pz*Dataset.Xd;

        [beta,ci,xi,~,stats] = regress(mean_utility,Xd);

        result.demand.xi = xi;
        result.demand.beta = beta;
        result.demand.alpha = beta(end);
        result.demand.ci = ci;
        result.demand.stats = stats;

        [b, e] = get_markup(Dataset, params, result);

        Dataset.mean_utility = mean_utility;
        result.demand.elasticities = e;

    otherwise

        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Step 1: Estimation
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
32          %================  1.1: Matrix of Theta
                ============================
33          %theta=[alpha; sigma_p; lambda]
34          nb_init = params.nb_init;
35          nK = params.nK;
36
37          % Set particular inital values
38          theta_start= params.theta_start;
39
40          % Set bounds
41          lower_bound = params.lower_bound;
42          upper_bound =params.upper_bound;
43
44          theta_init = linspace(lower_bound,upper_bound, nb_init);
45
46          % Combine inital value and random grid points
47          theta_init=[theta_start, theta_init];
48
49
50          %================  1.2. Optimal GMM
                ============================
51          options = optimoptions('fmincon','Display','iter', ...
52              'SpecifyObjectiveGradient',false,'CheckGradients',
                    false, ...
53              'FiniteDifferenceType', 'central', ...
54              'FiniteDifferenceStepSize', 1e-10,'StepTolerance',1e
                    -10, ...
55              'MaxFunEval', inf, 'MaxIter', inf);
56
57          %%%%%% Find estimates using different initial values
58          theta_mat = zeros(nb_init, nK);
59          for i=1:params.nb_init
60              % Estimation
61              [theta_mat(i),~,~,~,~,hessian]  = fmincon('gmm_logit
                    ', theta_init(i), [], [], [], [], ...
62                  lower_bound, upper_bound, [], options, ...
63                  Dataset, params);
64
65              params.hessian = hessian;
66              [fval, est] = gmm_logit(theta_mat(i), Dataset, params)
                    ;
67              result{i}= est;
68              result{i}.start = theta_init(i);
69              result{i}.theta = theta_mat(i);
70              result{i}.fval = fval;
```

```
            clc;
        end

end

end
```

vertical_model

```
function [result, Dataset] = vertical_model(Dataset, params)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
lambda = params.lambda;
specification = params.specification;

shares = Dataset.shares;
price = Dataset.data.price;

% Add outside option price and share
shares = [1-sum(shares); shares];
price = [0; price];

% Compute mean utility for alpha ~ exponential
share_cumsum = cumsum(shares);
price_diff = diff(price);

mean_utility = -log(share_cumsum(1:end-1,1)).*price_diff/lambda;
mean_utility = cumsum(mean_utility);

switch specification
    case 'demand'
        % OLS of mean_utility

        Xd = Dataset.Pz*Dataset.Xd;
        [beta,ci,xi,~,stats] = regress(mean_utility,Xd);

        [b, e] = get_markup(Dataset, params);
        result.demand.elasticities = e;

        Dataset.mean_utility = mean_utility;
        result.demand.elasticities = e;
        result.demand.xi = xi;
        result.demand.beta = beta;
        result.demand.ci = ci;
        result.demand.stats = stats;
```

```
37
38      otherwise
39          Xd = Dataset.Pz*Dataset.Xd;
40          Xs = Dataset.Pz*Dataset.Xs;
41
42          [beta,ci_d,xi,~,stats_d] = regress(mean_utility,Xd);
43
44          result.demand.xi = xi;
45          result.demand.beta = beta;
46          result.demand.ci = ci_d;
47          result.demand.stats = stats_d;
48
49          [b, e] = get_markup(Dataset, params);
50
51          marginal_cost = Dataset.data.price-b;
52
53          [gamma,ci_s,wi,~,stats_s] = regress(marginal_cost,Xs);
54
55          result.supply.markup = b;
56          result.supply.elasticities = e;
57          result.supply.wi = wi;
58          result.supply.gamma = gamma;
59          result.supply.ci = ci_s;
60          result.supply.stats = stats_s;
61
62  end
63
64  end
```

prepare_data

```
1
2  %% Initialize variables.
3  filename = 'Yr18_PSet_BLP_data_no_header.txt';
4  delimiter = '\t';
5
6  %% Read columns of data as text:
7  % For more information, see the TEXTSCAN documentation.
8  formatSpec = '%s%s%s%s%s%s%[^\n\r]';
9
10 %% Open the text file.
11 fileID = fopen(filename,'r');
12
13 %% Read columns of data according to the format.
14 % This call is based on the structure of the file used to generate
```

```
    this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric text to numbers.
% Replace non-numeric text with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = mat2cell(dataArray{col}, ones(length(dataArray{col}), 1));
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,4,5,6]
    % Converts text in the input cell array to numbers. Replaced non-numeric
    % text with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1)
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData(row), regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if numbers.contains(',')
                thousandsRegExp = '^[-/+]*\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(numbers, thousandsRegExp, 'once'))
                    numbers = NaN;
```

```
48                      invalidThousandsSeparator = true;
49                  end
50              end
51              % Convert numeric text to numbers.
52              if ~invalidThousandsSeparator
53                  numbers = textscan(char(strrep(numbers, ',', '')),
                        '%f');
54                  numericData(row, col) = numbers{1};
55                  raw{row, col} = numbers{1};
56              end
57          catch
58              raw{row, col} = rawData{row};
59          end
60      end
61  end
62
63
64  %% Create output variable
65  Dataset.data = table;
66  Dataset.data.price = cell2mat(raw(:, 1));
67  Dataset.data.quantity = cell2mat(raw(:, 2));
68  Dataset.data.weight = cell2mat(raw(:, 3));
69  Dataset.data.hp = cell2mat(raw(:, 4));
70  Dataset.data.AC = cell2mat(raw(:, 5));
71  Dataset.data.firm = categorical(cell2mat(raw(:, 6)));
72
73  %% Clear temporary variables
74  clearvars filename delimiter formatSpec fileID dataArray ans raw
      col numericData rawData row regexstr result numbers
      invalidThousandsSeparator thousandsRegExp;
75
76  save input/Dataset
```