Advanced Industrial Organization II
Problem Set 2

Eliot Abrams　　Hyinmin Park　　Alexandre Sollaci

February 9, 2016

Question 1

If the utility of consumer i when buying product j is given by

$$u_{i,j} = \alpha_i(w_i - p_j) + \mathbf{x}_j' \beta_i + \xi_j + \varepsilon_{i,j}$$

while having the utility of the normalized good normalized to zero, then including a constant term – the first column of $\mathbf{x}_j$ is composed of 1's for every j – then the term $\beta_i^{(1)}$ can be interpreted as the utility that consumer i gets from consuming the "inside good". For example, if we are estimating demand for cars and the "outside good" is not purchasing a car at all, then $\beta_i^{(1)}$ will capture the utility that a consumer enjoys when buying a car, regardless of its characteristics. Note that $\beta_i^{(1)}$ may vary between individuals, since the value of a car for different people may depend on their own demographic characteristics (this is the approach we follow in our specification of the BLP below).

Question 2

Our estimates for the simple model for α and β are

$$\hat{\alpha} = 1.8185 \quad \text{and} \quad \hat{\beta} = \begin{bmatrix} 0.9638 \\ 1.4510 \\ 1.7228 \\ 0.3709 \end{bmatrix}$$

Since we follow the MPEC approach, we readily have an estimate for the vector of structural errors, ξ, given as part of the output of the GMM problem. It is then straightforward to compute the optimal GMM matrix as

$$\hat{W} = \frac{1}{J} \left[Z' \hat{D}_\xi Z \right]^{-1}$$

where J is the number of products$\times$markets and

$$\hat{D}_\xi = \begin{pmatrix} \hat{\xi}_1^2 & 0 & \dots & 0 \\ \vdots & \hat{\xi}_2^2 & \ddots & \vdots \\ 0 & 0 & \dots & \hat{\xi}_J^2 \end{pmatrix}$$

A brief note on the estimation procedure

The program we wrote to estimate the logit model was written in Julia and takes advantage of the JuMP (Julia for Mathematical Programming) package and the Ipopt solver.

We chose Julia because it is a compiled language – and thus faster – but which retains some of the flexibility of interpreted languages. Furthermore, JuMP saves us the work of having to provide derivatives to Ipopt when solving the GMM minimization problem, while still being able to quickly compute a solution.

The combination of Julia and JuMP worked well in the simple logit model. However, the full BLP problem proved to be much more demanding, taking the "regular" version of JuMP several days of computation while still not arriving at a solution. While trying to fix the problem we were fortunate enough to be able to talk to the developers of JuMP, who informed us that the code for the package had recently been rewritten and that we should try to use its development version to solve the model. This is simply done by running the following commands:

```
Pkg.checkout("JuMP")
Pkg.update()
```

When this also failed (Ipopt reported having converged to a point of local infeasibility), we switched solver from Ipopt to KNITRO. This as also a suggestion of the developers of JuMP, arguing that KNITRO is a better non-linear solver. The results we present here are based on this last approach (Julia + JuMP + KNITRO); we haven't got the solution to converge yet, but, on the other hand, the program has not returned any errors and the parameters seem to evolve nicely with each iteration.

Separately, we have also tried working directly with Ipopt (without using JuMP) to solve the BLP model. This separate approach involved manually coding the first derivatives of the GMM minimization problem suggested by MPEC to the BLP. Unfortunately, we could not get it to work, as Ipopt kept returning a segmentation error.

Question 3

Let K be the number of columns in $\mathbf{x}$ (in this particular case, $K = 4$). We will estimate $1 + K + 3(K + 1)$ parameters, since

- α is a 1×1 vector – or just a scalar;
- β is a $K \times 1$ vector, the same dimension as $\mathbf{x}$;
- $\mathbf{\Pi}$ has two $(K+1)\times 1$ blocks: Π_{inc} and Π_{age}, referring to income and age of each consumer. In the way we set up our model, the first K entries of both Π_{inc} and Π_{age} will interact with product characteristics, $\mathbf{x}$, in the market-share equation, while the $K + 1^{th}$ entry will interact with price, $\mathbf{p}$.
- $\sqrt{\Sigma}$ has $K \times 1$ parameters, since we assume that Σ is a diagonal matrix. As before, in the way we set up our model, $\sigma_1, \ldots, \sigma_K$ will interact with characteristics while σ_{K+1} will interact with prices in the market-share equation.

Having said that, we can define the market-share in the model by

$$s_j = \frac{1}{N}\sum_{n=1}^{N} \frac{\exp\left(\sum_{k=1}^{K} x_{jk} Y_{k,n}^{x} - Y_n^{p} p_j + \xi_j\right)}{1 + \sum_{j=1}^{J} \exp\left(\sum_{k=1}^{K} x_{jk} Y_{k,n}^{x} - Y_n^{p} p_j + \xi_j\right)}$$

where

$$Y_{k,n}^{x} = \beta^{(k)} + \Pi_{\text{inc}}^{(k)}\text{inc}_n + \Pi_{\text{age}}^{(k)}\text{age}_n + \sigma_k \nu_n^k$$

and

$$Y_n^p = \alpha + \Pi_{\text{inc}}^{(K+1)} \text{inc}_n + \Pi_{\text{age}}^{(K+1)} \text{age}_n + \sigma_{K+1} \nu_n^{K+1}.$$

Question 4

As pointed out in the discussion above, out code has not converged to a solution for the BLP model. The estimates we return here are our best guess after letting the program run for XXXX iterations. Those values are

$$\hat{\alpha} = -0.8699, \quad \hat{\beta} = \begin{bmatrix} 0.9638 \\ 2.6917 \\ 1.0280 \\ 1.3045 \end{bmatrix}, \quad \hat{\Pi}_{\text{inc}} = \begin{bmatrix} 1.3283e^{-7} \\ 0.2348 \\ 0.7485 \\ -0.3961 \\ 0.3898 \end{bmatrix}, \quad \hat{\Pi}_{\text{age}} = \begin{bmatrix} 3.9302e^{-7} \\ 0.0352 \\ -0.0503 \\ 0.0090 \\ 0.1638 \end{bmatrix}, \quad \hat{\sigma} = \begin{bmatrix} -4.0364e^{-12} \\ -0.9308 \\ -0.1276 \\ -1.7015 \\ -1.5043 \end{bmatrix}$$

Comparing the full BLP estimates with the simple logit estimates, we can see that ....

Appendix: code

```
# 2016 Winter Advanced IO PS2
# Hyunmin Park, Eliot Abrams, Alexandre Sollaci

#=
julia_implementation_of_blp.jl

Julia code for implementing a BLP model using MPEC to solve for parameters
=#

#####################
##      Setup      ##
#####################

using Ipopt
using KNITRO
using JuMP
using DataFrames
EnableNLPResolve()


#####################
##      Data       ##
#####################

# Load data
product = DataFrames.readtable("dataset_cleaned.csv", separator = ',',
 header = true);
population = DataFrames.readtable("population_data.csv", separator = ',',
 header = true);

# Define variables
x = product[:,3:6];
p = product[:,7];
z = product[:,8:13];
s0 = product[:,14];
s = product[:,2];
iv = [x z];
inc = population[:,1];
age = population[:,2];
v = population[:,3:7];

# Store dimensions
K = size(x,2);
L = K+size(z,2);
J = size(x,1);
N = size(v,1);
M = size(v,2);


##########################
##  Simple Logit Model  ##
##########################

# Setup the simple logit model
logit = Model(solver = IpoptSolver(tol = 1e-8, max_iter = 1000,
output_file = "logit.txt"));
```

```
# Define variables
@defVar(logit, g[1:L]);
@defVar(logit, xi[1:J]);
@defVar(logit, alpha);
@defVar(logit, beta[1:K]);

# We minimize the gmm objective with the identity as the weighting matrix
# subject to the constraints g = sum_j xi_j iv_j and market share equations
@setObjective(logit, Min, sum{g[l]^2,l=1:L});
@addConstraint(
    logit,
    constr[l=1:L],
    g[l]==sum{xi[j]*iv[j,l], j=1:J}
);
@addNLConstraint(
    logit,
    constr[j=1:J],
    xi[j]==log(s[j])-log(s0[j])+alpha*p[j]-sum{beta[k]*x[j,k],k=1:K}
);

# Solve the model
status = solve(logit);

# Print the results
println("alpha = ", getValue(alpha))
println("beta = ", getValue(beta[1:K]))

# Save results to use in the setup of BLP Model
g_logit=getValue(g);
xi_logit=getValue(xi);
alpha_logit=getValue(alpha);
beta_logit=getValue(beta);


##########################
##      BLP Model       ##
##########################

# Calculate the optimal weighting matrix
iv = convert(Array, iv)
W = inv((1/J)*iv'*Diagonal(diag(xi_logit*xi_logit'))*iv);

# Setup the BLP model
BLP = Model(solver = KnitroSolver(KTR_PARAM_HESSOPT=6,
KTR_PARAM_OUTMODE=2, KTR_PARAM_LINSOLVER=5, KTR_PARAM_MAXIT=100));

# Defining variables - set initial values to estimates from the logit model
@defVar(BLP, g[x=1:L], start=(g_logit[x]));
@defVar(BLP, xi[x=1:J], start=(xi_logit[x]));
@defVar(BLP, alpha, start=alpha_logit);
@defVar(BLP, beta[x=1:K], start=beta_logit[x]);

# Defining variables - heterogeneity parameters
@defVar(BLP, piInc[1:K+1]);
@defVar(BLP, piAge[1:K+1]);
@defVar(BLP, sigma[1:K+1]);
```

```
# We minimize the gmm objective - using the optimal weighting matrix!
# subject to g = sum_j xi_j iv_j and market share equations -
# Note that where we assign each shock could have minor effect on estimation results
# shock 1 : taste shock to x1
# shock 2 : taste shock to x2
# shock 3 : taste shock to x3
# shock 4 : taste shock to x4
# shock 5: taste shock to price
@setObjective(BLP, Min,sum{sum{W[i,j]*g[i]*g[j],i=1:L},j=1:L});
@addConstraint(
    BLP,
    constr[l=1:L],
    g[l]==sum{xi[j]*iv[j,l],j=1:J}
);
@defNLExpr(
    BLP,
    denom[n=1:N],
    sum{
        exp(beta[1]
            -(alpha+piInc[K+1]*inc[n]+piAge[K+1]*age[n]+sigma[K+1]*v[n,K+1])*p[h]
            +sum{(beta[k]+piInc[k]*inc[n]+piAge[k]*age[n]+sigma[k]*v[n,k])*x[h,k],
             k=1:K} +xi[h] ) , h=1:J}
);
@addNLConstraint(
    BLP,
    constr[j=1:J],
    s[j]==(1/N)*
        sum{
            exp(beta[1]
                -(alpha+piInc[K+1]*inc[n]+piAge[K+1]*age[n]+sigma[K+1]*v[n,K+1])*p[j]
                 +sum{(beta[k]+piInc[k]*inc[n]+piAge[k]*age[n]+sigma[k]*v[n,k])*x[j,k],
                k=1:K} +xi[j])/denom[n] ,n=1:N}
);

# Solve the model
status = solve(BLP);

# Print the results
println("alpha = ", getValue(alpha))
println("beta = ", getValue(beta[1:K]))
println("piInc = ", getValue(piInc[1:K+1]))
println("piAge = ", getValue(piAge[1:K+1]))
println("sigma = ", getValue(sigma[1:K+1]))
```