IO Homework

December 27, 2020

Done by: Wenxuan Xu and Andrei Zaloilo

1 Results

1.1 Q1: Standard Nash-Bertrand Competition

We use FOCs directly to calculate the prices: given shares, prices are chosen to set the FOCs equal to zero, and then the shares and corresponding derivatives are adjusted to new prices. Process conveges to the unique value irrespective of the initial guess.

Result: Prices≈(1.80,1.97,2.15,2.35,2.56), increasing in quality. Shares are decreasing in quality, and about 49% of all consumers pick the outside good.

1.2 Q.2: equilibrium with two-part tariffs

No disribution costs and wholesale price=production cost implies that the model is equivalent to one where firm 1 is the producer of goods 1 and 2, and firm 2 is the producer of goods 3,4, and 5.

Numerically, this problem is exactly the same as Q.1 except for different FOCs, since each firm now controls more than one good.

Result: Prices≈(2.16,2.34,2.70,2.94,3.18), monotonically increasing in quality, larger than the standard competition case in the previous question. Shares are monotonically decreasing in quality, and about 56% of the customers pick the outside good.

1.3 Question 3: Welfare change

Result: welfare has decreased by about 19% from its original value, as a result of increased power of firms.

2 Code

2.0.1 Setup

[138]:
```
using Random, Distributions, Printf
Random.seed!(2020); # Setting the seed
```

2.0.2 Calculating Demand

$$u_{ij} = 4 + j/5 - \alpha_i p_j + \epsilon_{ij}$$

Thus, given α_i,

$$prob(i,j) = \frac{exp(4 + j/5 - \alpha_i p_j)}{\sum_{j=1}^{5}[exp(4 + j/5 - \alpha_i p_j)] + 1} = num_{ij}/denom_i$$

Then, given α_i,

$$\frac{\partial s_j}{\partial p_j} = \sum_{i=1}^{2000} -\alpha_i \frac{num_{ij}(denom_i - num_{ij})}{denom_i^2}$$

and

$$\frac{\partial s_k}{\partial p_j} = \sum_{i=1}^{2000} \alpha_i \frac{num_{ij} num_{ik}}{denom_i^2}$$

[139]:
```
const epsp=1e-8;
d=Normal(1.0,1.0);
logalpha=rand(d,2000);
alpha=exp.(logalpha);
```

[140]:
```
function SHARES(p::Array{Float64}) #Calculates market shares and derivatives
sharedenom=zeros(2000);
share=zeros(5);
sumshare=0;
sharederiv=zeros(5);
sharederivk=zeros(5,5);

#Denominator of the share function
for i=1:2000
for j=1:5
sharedenom[i]=sharedenom[i]+exp(4+j/5-alpha[i]p[j]);
    end
sharedenom[i]=sharedenom[i]+1;
    end

#Shares
for j=1:5
for i=1:2000
share[j]=share[j]+exp(4+j/5-alpha[i]p[j])/sharedenom[i];
        end
share[j]=share[j]/2000
    end

#Sum of shares
for j=1:5
sumshare=sumshare+share[j];
    end
```

```
#Derivative ds_j/dp_j
for j=1:5
for i=1:2000
sharederiv[j]=sharederiv[j]-alpha[i]*exp(4+j/
 ↪5-alpha[i]p[j])*(sharedenom[i]-exp(4+j/5-alpha[i]p[j]))/(sharedenom[i]^2);
    end
sharederiv[j]=sharederiv[j]/2000 #Do I need to normalize this?
    end

#Derivative ds_k/dp_j

for k=1:5
for j=1:5
for i=1:2000
sharederivk[k,j]=sharederivk[k,j]+alpha[i]*exp(4+k/5-alpha[i]p[k])*exp(4+j/
 ↪5-alpha[i]p[j])/(sharedenom[i]^2);
    end
sharederivk[k,j]=sharederivk[k,j]/2000
if (j==k)
    sharederivk[k,j]=sharederiv[j];
            end
    end
    end


return(share,sharederiv,sharedenom,sharederivk,sumshare)
end
```

[140]:
```
SHARES (generic function with 1 method)
```

2.1 Q1: Standard Nash-Bertrand Competition

[150]:
```
#Calculating Nash-Bertrand equilibrium for Q.1
function Q1(print::Int64)
p=zeros(5);
share=zeros(5)
sharederiv=zeros(5);
sharederivk=zeros(5);
critpj=zeros(5);
sumshare=0;
logsum=zeros(2000);
utility=zeros(2000,5);
welfare1=0;
critp=1.0;
```

```
epsfoc=1e-4;

#Initial guess for prices.
    for j=1:5
p[j]=2000
    end


while critp>epsp
pold=copy(p);
share=SHARES(p)[1]
sharederiv=SHARES(p)[2]
sharedenom=SHARES(p)[3]
sharederivk=SHARES(p)[4]
sumshare=SHARES(p)[5]
#Prices chosen to satisfy FOCs
for j=1:5
p[j]=1+j/8-share[j]/sharederiv[j]

critpj[j]=abs.(p[j]-pold[j])
p[j]=0.9*p[j]+0.1pold[j]
    end
critp=maximum(critpj);
end

#Sanity check
if (abs(p[3]-(1+3/8-share[3]/sharederiv[3]))>epsfoc)
@printf("%8s\n","Sanity check failed")
end

if (print==1)
@printf("%8s\n","Prices");
@printf("%5f %5f %5f %5f %5f\n",p[1],p[2],p[3],p[4],p[5]);

@printf("%8s %8s %8s %8s %8s %8s \n","Shares","","","","","Sum");
@printf("%5f %5f %5f %5f %5f %5f␣
 ↪\n",share[1],share[2],share[3],share[4],share[5],sumshare);
 end

#Calculating welfare using logsum formula

for i=1:2000
for j=1:5
utility[i,j]=exp(4+j/5-alpha[i]p[j]);
        end
    end
```

```
for i=1:2000
logsum[i]=log(1+utility[i,1]+utility[i,2]+utility[i,3]+utility[i,4]+utility[i,5]);
 ↪
    end

for i=1:2000
welfare1=welfare1+logsum[i]/2000
    end

@printf("%8s\n","Welfare 1")
@printf("%5f \n",welfare1)

return(welfare1);
end

Q1(1);
```

```
  Prices
1.802948 1.973153 2.154584 2.348172 2.555125
  Shares                                                Sum
0.120985 0.110091 0.100869 0.093121 0.086637 0.511703
Welfare 1
1.708466
```

2.2 Q.2: equilibrium with two-part tariffs

[152]:
```
#Calculating Nash-Bertrand equilibrium for Q.2.
function Q2(print::Int64)
p=zeros(5);
share=zeros(5)
sharederiv=zeros(5);
sharederivk=zeros(5);
sumshare=0;
critpj=zeros(5);
logsum=zeros(2000);
utility=zeros(2000,5);
welfare2=0;
critp=1.0;
epsfoc=1e-4;


#Initial guess for prices
    for j=1:5
p[j]=2000
    end
```

```
while critp>epsp
pold=copy(p);

share=SHARES(p)[1]
sharederiv=SHARES(p)[2]
sharedenom=SHARES(p)[3]
sharederivk=SHARES(p)[4]
sumshare=SHARES(p)[5]

#Adjusting prices to satisfy FOCs. Robust to sequencing!
p[1]=1+1/8-((p[2]-1-2/8)*sharederivk[2,1]+share[1])/sharederiv[1]
p[2]=1+2/8-((p[1]-1-1/8)*sharederivk[2,1]+share[2])/sharederiv[2]
p[3]=1+3/8-((p[4]-1-4/8)*sharederivk[4,3]+(p[5]-1-5/
 ↪8)*sharederivk[5,3]+share[3])/sharederiv[3]
p[4]=1+4/8-((p[3]-1-3/8)*sharederivk[4,3]+(p[5]-1-5/
 ↪8)*sharederivk[5,4]+share[4])/sharederiv[4]
p[5]=1+5/8-((p[3]-1-3/8)*sharederivk[3,5]+(p[4]-1-4/
 ↪8)*sharederivk[4,5]+share[5])/sharederiv[5]

for j=1:5
critpj[j]=abs.(p[j]-pold[j])
p[j]=0.9*p[j]+0.1pold[j]
end

critp=maximum(critpj);
end

#Sanity check
if (abs(p[3]-(1+3/8-((p[4]-1-4/8)*sharederivk[4,3]+(p[5]-1-5/
 ↪8)*sharederivk[5,3]+share[3])/sharederiv[3]))>epsfoc)
@printf("%8s\n","Sanity check failed")
end


if (print==1)
@printf("%8s\n","Prices");
@printf("%5f %5f %5f %5f %5f\n",p[1],p[2],p[3],p[4],p[5]);
@printf("%8s %8s %8s %8s %8s %8s \n","Shares","","","","","Sum");
@printf("%5f %5f %5f %5f %5f %5f␣
 ↪\n",share[1],share[2],share[3],share[4],share[5],sumshare);
    end

#Calculating welfare using logsum

for i=1:2000
for j=1:5
utility[i,j]=exp(4+j/5-alpha[i]p[j]);
```

```
        end
    end

for i=1:2000
logsum[i]=log(1+utility[i,1]+utility[i,2]+utility[i,3]+utility[i,4]+utility[i,5]);
 ↪
    end

for i=1:2000
welfare2=welfare2+logsum[i]/2000
    end

@printf("%8s\n","Welfare 2")
@printf("%5f \n",welfare2)


return(welfare2);
end

Q2(1);
```

```
  Prices
2.155003 2.344767 2.701046 2.939235 3.179711
  Shares                                                Sum
0.114357 0.104854 0.078872 0.072119 0.067398 0.437600
Welfare 2
1.392319
```

2.3 Question 3: Welfare change

[143]:
```
function Q3()
welfare1=Q1(0);
welfare2=Q2(0);
welfarechange=welfare2-welfare1;
welfarechangepcnt=welfarechange/welfare1

@printf("%8s\n","Change in welfare")
@printf("%8s %8s\n","Absolute","Percent")

@printf("%5f %5f\n",welfarechange,welfarechangepcnt)
end

Q3();
```

```
Welfare 1
1.708466
Welfare 2
```

```
1.392319
Change in welfare
Absolute  Percent
-0.316148 -0.185048
```