HW2 Report

Ece Teoman

September 26, 2018

Here, I include the code and the output but the m.files and the diary can be found in the directory.

1

The resulting demands for the given parameterization are as follows:

```
D_A =

0.4223

D_B =

0.4223

D_0 =

0.1554
```

2

Given $q_A = q_B = 2$, solving for the Nash pricing equilibrium by using Broyden method results in the following:

```
sol_broyden =

1.5989      1.5989



Broyden_runtime =

0.0284
```

3

Under the same parameter values, now using Gauss-Seidel method to solve for the pricing equilibrium results in the following:

```
solution_Gauss_Seidel =

1.5989
1.5989



Gauss_Seidel_runtime =

0.3158
```

In this case, Broyden was faster than Gauss-Seidel.

4

Solving for the equilibrium using the given update rule results in:

```
solution_update =


1.5989
1.5989


update_runtime =


0.0048
```

It converges and this method is the fastest so far.

5

Given the parameters, the resulting pricing equilibrium, by using Gauss-Seidel method is:

```
p_a =

Columns 1 through 12

1.8671    1.8465    1.8239    1.7995    1.7735    1.7461    1.7176
1.6883    1.6586    1.6287    *1.5989*    1.5697

Columns 13 through 16

1.5411    1.5134    1.4867    1.4612

p_b =

Columns 1 through 12

1.1481    1.1743    1.2041    1.2378    1.2757    1.3179    1.3646
1.4160    1.4721    1.5331    *1.5989*    1.6697

Columns 13 through 16

1.7453    1.8257    1.9109    2.0008
```

So the equilibrium prices are the same as in the previous parts. The resulting graph is as follows:

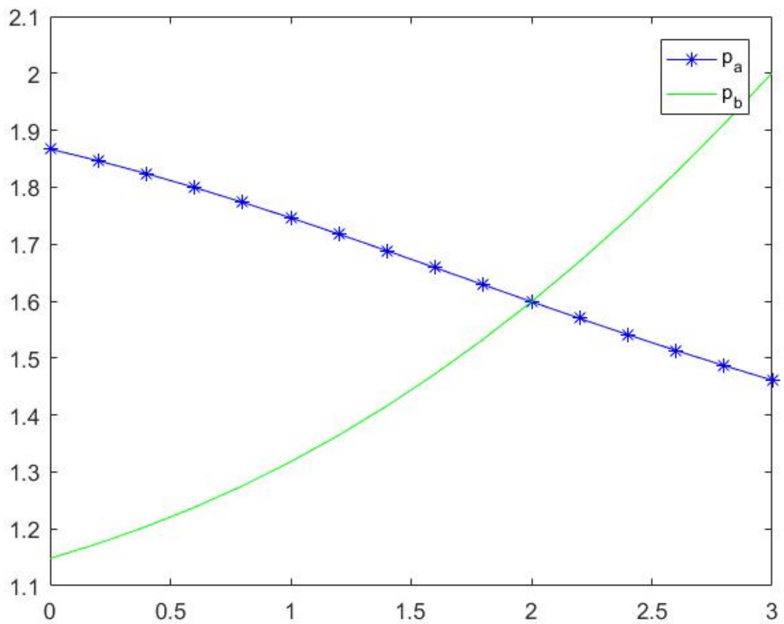

Figure 1:

6 MATLAB codes

Here are the functions and the rest of the code I've written to get the answers above:

6.1 Functions

```
function [D_A, D_B, D_0 ] = hw2_q1( p_A, p_B, q_A, q_B )
D_A = exp(q_A - p_A)/[1+exp(q_A - p_A)+exp(q_B - p_B)];
D_B = exp(q_B - p_B)/[1+exp(q_A - p_A)+exp(q_B - p_B)];
D_0 = 1/[1+exp(q_A - p_A)+exp(q_B - p_B)];
%D_A is the demand func for good A
%D_B is the demand func for good B
%D_0 is the demand func for the outside option
```

```
end

function F = foc( q_A, q_B, p )
D_A = exp(q_A - p(1))/[1+exp(q_A - p(1))+exp(q_B - p(2))];
D_B = exp(q_B - p(2))/[1+exp(q_A - p(1))+exp(q_B - p(2))];
F= [1 - p(1)*(1-D_A); 1 - p(2)*(1-D_B)];
%F is first-order conditions

end

function X = onestep( q_A, q_B, p )
options = optimset('Display','off');
foc_A= @(p_A) 1- p_A*(1- (exp(q_A - p_A)/[1+exp(q_A - p_A)+exp(q_B - p(2))]));
res_A = fsolve(foc_A, p(1), options);

foc_B= @(p_B) 1- p_B*(1- (exp(q_B - p_B)/[1+exp(q_B - p_B)+exp(q_A - p(1))]));
res_B = fsolve(foc_B, p(2), options);



X= [res_A; res_B];

%res_j is the solution of j's FOC



end

function x = price_update( q_A, q_B, p )
D_A= exp(q_A - p(1))/[1+exp(q_A - p(1))+exp(q_B - p(2))];
D_B= exp(q_B - p(2))/[1+exp(q_A - p(1))+exp(q_B - p(2))];
x= [1/(1-D_A); 1/(1-D_B)];
```

```
end
```

6.2 Main

```
%% Q1
[D_A, D_B, D_0 ] = demand(1,1,2,2)


%% Q2
q=[2 2]';
optim_p=@(p) foc(q(1),q(2),p);
inguess=ones(2,1);


tol=1e-15;


optset('broyden','tol',tol);
tic;
solution_broyden = broyden(optim_p,inguess)'
Broyden_runtime= toc



%% Q3
p_0= ones(2,1);
error= 1;
iter= 1;
tic;


while abs(error)>=tol && iter <10000
```

```
    x= onestep(2, 2, p_0);
    error= norm(p_0 - x);
    p_0= x;
    iter= iter+1;
end
solution_Gauss_Seidel= p_0
Gauss_Seidel_runtime= toc

%% Q4

p_0= ones(2,1);
error= 1;
iter= 1;
tic;
while abs(error)>=tol && iter<10000
    x= price_update(2, 2, p_0);
    error= norm(p_0 - x);
    p_0= x;
    iter= iter+1;
end

solution_update= p_0
update_runtime= toc

%% Q5
p_0= ones(2,1);
price= zeros(2,16);
error= 1;
```

```
iter= 1;
foriter=0;
tic;
for q_B=0:0.2:3
    foriter=foriter+1;
    p_0= ones(2,1);
    error=1;
while abs(error)>=tol && iter<10000

    x= onestep(2, q_B, p_0);
    error= norm(p_0 - x);
    p_0= x;
    iter= iter+1;
end
price(: , foriter)= p_0;
end

q_b= 0:0.2:3;
p_a= price(1, :)
p_b= price(2, :)
plot ( q_b , p_a , 'b*-', q_b, p_b, 'g');
legend ('p_a', 'p_b');
```