Computational and Empirical Methods

Assignment VI

Instructor: Prof. Gunnar Heins

Teaching Assistant: Pedro de Sousa Almeida

Student: C.H. (Chenhui Lu)

UFID: 76982846

 

```
%%%%%%%%%%%%%%%%%%%%
%   Assignment 6   %
%%%%%%%%%%%%%%%%%%%%

clear                % Empty the Workspace
clc                  % Clear the Screen
format compact       % compact format

%% 1 Cournot - Linear Case (15 points)

%In a Cournot oligopoly, all  rms produce and sell a homogeneous product. All  rm's↙
output
%is sold in a single market, for which there is a speci ed market demand. Each  rm↙
chooses its
%own quantity to maximize its own pro ts, but cannot directly infuence the quantity↙
produced
%by its rivals. Here, we consider a Cournot oligopoly with three  rms that is↙
characterized by
%the following inverse market demand function and individual  rm cost function,↙
respectively:
%P(Q) = 450 - 0.125Q
%Ci(qi) = 50qi + 180

%When each  rm maximizes its own pro t by selecting its own quantity, taking their↙
rivals'
%quantities as given, we obtain the following First Order Conditions:
%450 - 0.25q1 - 0.125q2 - 0.125q3 - 50 = 0
%450 - 0.125q1 - 0.25q2 - 0.125q3 - 50 = 0
%450 - 0.125q1 - 0.125q2 - 0.25q3 - 50 = 0

%% 1.1. Present the linear system of First Order Conditions in matrix form.
%need to rewrite the equitions as Aq = b
% - 0.25q1 - 0.125q2 - 0.125q3 = -400
% - 0.125q1 - 0.25q2 - 0.125q3 = -400
% - 0.125q1 - 0.125q2 - 0.25q3 = -400

%[-0.25, -0.125, -0.125; -0.125, -0.25, -0.125; -0.125, -0.125, -0.25] *
%[q1; q2; q3] = [-400; -400; -400]

%% 1.2. Solve it in Matlab via Matrix Inversion.
A = [-0.25, -0.125, -0.125; -0.125, -0.25, -0.125; -0.125, -0.125, -0.25];  % set up A↙
metrix
b = [-400; -400; -400];                                                     % set up b↙
metrix
q = inv(A)*b;                                                                 % q = A^↙
(-1)b
q
%q =
%  800
%  800
%  800

%% 1.3. Can you apply the iterative method for this problem that we covered in class?
%Why or why not? If your answer is yes,  nd the three quantities via this approach.

%Yes, with Gauss - Seidel Method.

%Gauss-Jacobi Method  (could not solve the problem since couldnot update with new↙
```

```
paramiters, as shown below)

q1 = 300;                   % Initial guess
q2 = 300;
q3 = 300;

for i = 1:20
q1_new = (- 0.125*q2 - 0.125*q3 + 400)/0.25;     % Update q1
q2_new = (- 0.125*q1 - 0.125*q3 + 400)/0.25;     % Update q2
q3_new = (- 0.125*q1 - 0.125*q2 + 400)/0.25;     % Update q3
q1 = q1_new;
q2 = q2_new;
q3 = q3_new;
disp([q1, q2, q3]);
end

% 300; 300; 300

%Gauss-Jacobi Method, can solve the problem.
q1 = 300;             % Initial Guess
q2 = 300;
q3 = 300;

for i = 1:20
    q1 = (-0.125 * q2 - 0.125 * q3 + 400) / 0.25;   % Update
    q2 = (-0.125 * q1 - 0.125 * q3 + 400) / 0.25;
    q3 = (-0.125 * q1 - 0.125 * q2 + 400) / 0.25;

    disp([q1, q2, q3]);                              % show each result
end

%800; 800; 800


%% 2 Cournot - Nonlinear Case (20 points)
%In class we have covered the Cournot example with 2  rms (N = 2) and c = 2.
% In this question, you'll compute numerically how the number of firms N
% affects equilibrium prices and quantities. Speci cally, assume that there
% are N  rms, each producing a quantity qi. Total output in the economy is
% hence given by Q = q1 + q2 + ... + qN. Suppose the inverse demand for the
% good is given by
% P(Q) = 1 - Q
% and, as in class, each  firm's total cost equals
% TC = c*(2/3)*q^(3/2)
% As before, assume that parameter c = 2.

%% 2.1. Find the equilibrium quantity and price in the case of a monopolist
% instead, i.e. in the case of 1  rm. Use fsolve().

% Profit = Total Revenue-Total Cost
% Profit = Price * Total Profuction -Total Cost
% Profit  = (1-q)*q - 2*(2/3)*q^(3/2)

% first set up the function: HW6_Cournot

%calculation:
q0 = 1;                                    % Pick an initial guess
Cournot_handle = @(q) HW6_Cournot(q);      % Create the function handle = cricial↙
```

```
variable function(variable)
q_opt = fsolve(Cournot_handle, q0)        % Use fsolve  (hundle, initial guess)  to solve↙
the system

% Optimal production: Q_opt = 0.1340

p = 1 - q_opt
% eq price: p = 0.8660

%% 2.2. Find the equilibrium quantities and price in the case of 3 firms.
% P(Q) = 1 - q1 - q2 - q3
% TC = c*(2/3)*q^(3/2)
% Profit1  = (1 - q1 - q2 - q3)*q1 - 2*(2/3)*q1^(3/2)
% Profit2  = (1 - q1 - q2 - q3)*q2 - 2*(2/3)*q2^(3/2)
% Profit3  = (1 - q1 - q2 - q3)*q3 - 2*(2/3)*q3^(3/2)

% first set up the function: HW6_Cournot2

% calculation
q0 = [1;1;1];                                        % Pick an initial guess
Cournot_handle = @(q) HW6_Cournot2(q);      % Create the function handle = cricial↙
variable function(variable)
q_opt = fsolve(Cournot_handle, q0)       % Use fsolve  (hundle, initial guess)  to solve↙
the system
%q_opt = [0.0955, 0.0955, 0.0955]

p = (1 - q_opt(1) - q_opt(2) - q_opt(3))
% p = 0.7135

%% 2.3. Based on your results, what do you think will be the equilibrium price when N ->↙
infinity
%when N -> infinity, the market will become perfectly competitive market.
%Then the market price will equal the marginal cost : P = 1 - Q = MC = 2 *
%q^(1/2)


%% 3 Finding the Equilibrium (30 points)
%Suppose you have an economy with a demand for cars of
% x = 2 - p
% supply
% x = -1 + exp(p)
%such that in equilibrium it must hold that
%2 - p = -1 + exp(p)
% <==> 3 - p - exp(p) = 0

%% 3.1. Given that equilibrium quantities cannot be negative,
% in what range will the equilibrium price need to be?

%Since x >= 0
% 1. 2 - p >= 0,  2. -1 + exp(p) >= 0
% 0 <= p <= 2

%% 3.2. Use this range and do a grid search to find the equilibrium price.
% Use increments of 0.01.

p = meshgrid(0:0.01:2);                              % build up the grid

for i = 1:size(p,1)                                  % double loops
```

```
    for j = 1:size(p,2)
        diff(i,j) = 3 - p(i,j) - exp(p(i,j));    % function
        obj(i,j) = diff(i,j)^2;                  % objective is the diff^2
    end
end
c_min = min(min(obj));                           % find the minimum of obj, the ideal↙
minimum should colse to 0
[i_max, j_max] = find(obj == c_min,1);
solution = p(i_max,j_max)
%solution = 0.7900

%% 3.3. Now instead, use Newton's method as covered in class to find the equilibrium↙
price.
% Do not use fsolve(). Do you find the same price as you did with the grid search?
%3 - p - exp(p) = 0

p = 6                           % initial guess
for i = 1:20                    % iterate 20 times
    num = 3 - p - exp(p);       % Numerator is the original qeuition
    denom = -1 - exp(p);        % Denomerator is the FOC
    p = - num/denom + p;        % The iteration starts
    p
end

% p = 0.7921 the solution is different to what we got with grid search,
% This discrepancy is likely due to us only using a second-order Taylor series,
% which results in a loss of some accuracy.

%% 3.4. Now suppose you have two cars with demand functions
% 1x1 + 0x2 + 0.2p1 - 0.3p2 = 2
% 0x1 + 1x2 - 0.3p1 + 0.2p2 = 2
% and supply
% 1x1 + 0x2 - 1p1 + 0p2 = 1
% 0x1 + 1x2 + 0p1 - 1p2 = 1
%Set supply and demand equal for each type of car and solve for p1 and p2
% using matrix inversion.

%mertix:
A = [1,0,0.2,-0.3;0,1,-0.3,0.2;1,0,-1,0;0,1,0,-1];
b = [2;2;1;1];
x = A\b;

p1 = x(3)
p2 = x(4)

%solution: p1 = 1.1111, p2 = 1.1111.



%% 4 Pro t Maximization (35 points)

%Now, consider the problem of a firm that produces a good with production function
% F(K; L) = 3K^0.2 * L^0.4,
% using two inputs: Capital at price 2 and labor at price 1. Formally, the firm
% problem is to maximize profits  :
% max(K,L) = 3K^0.2 * L^0.4 - 2K - L

%% 4.1. Solve this problem numerically using a grid search,
```

```
% i.e. find the combination of K and L that maximizes prfits pai.
% Assume that K~[0,1] and L~[0,1] and use increments of 0.01.
% FOC on K: 0.6K^(-0.8)*L^0.4 - 2 = 0
% FOC on L: 1.2K^0.2*L^(-0.6) - 1 = 0
[K, L] = meshgrid(0:0.01:1,0:0.01:1);          % set the grid

for i = 1:size(K,1)                              % set double loops
    for j = 1:size(K,2)
        pai1(i,j) = 0.6*K(i,j)^(-0.8)*L(i,j)^0.4 - 2;  % profit function
        pai2(i,j) = 1.2*K(i,j)^0.2*L(i,j)^(-0.6) - 1;
        obj(i,j) = pai1(i,j)^2 + pai2(i,j)^2;
    end
end

c_min = min(min(obj));                           % find the maxmium of profit
[i_min,j_min] = find(obj == c_min);
solution = [K(i_min,j_min);L(i_min,j_min)]

%solution =   0.2000    0.8000

%% 4.2. Instead of using a grid search you can also derive the first-order conditions and
solve those:
% 0.6K^(-0.8)*L^0.4 = 2
% 1.2K^0.2*L^(-0.6) = 1
% Based on the discussions in class, what would be the most efficient way
% to solve these FOCs for K and L? Why?

%% 4.2.1. if we are asked to solve the maximization:
% Solution: Since we are doing maximization. Trust-Region Algorithm, which implies
Newton's Method. Newton's Method has very good
% convergence properties and comparative fast. In addition, computing Gradient and
Hessian manually
% and supplying it to the optimizer can substantially enhance the optimization.

%% 4.2.2. if we are asked to only solve this 2 equitions:
% soultion: The Newton' Method is the best, since the equitions are not
% linear.


%% 4.3. Solve the system using the method you proposed in question (2).
%% 4.3.1. if we are asked to solve the maximization:
I0 = [1;1];                                    % Pick an initial guess
objective_handle = @(I) max4_2(I);             % Create the function handle
options = optimoptions('fminunc','Algorithm','trust-region', 'GradObj', 'on', 'Hessian',
'on');
fminunc(objective_handle, I0, options)
% ans = 0.1972; 0.7887

%% 4.3.2. if we are asked to only solve this 2 equitions:
% 0.6K^(-0.8)*L^0.4 = 2
% 1.2K^0.2*L^(-0.6) = 1

% Pick an initial guess
K = 0.2;
L = 0.8;
I = [K; L];

for i = 1:10
```

```
F = [0.6*K^(-0.8)*L^0.4-2; 1.2*K^0.2*L^(-0.6)-1];              % Function
DF = [-0.48 * K^(-1.8) * L^(0.4), 0.24 * K^(-0.8) * L^(-0.6); 0.24 * K^(-0.8) * L^(-0.6),↙
-0.72 * K^(0.2) * L^(-1.6)];          % Gradient
I = - inv(DF) * F + I;                                                        % Newton↙
Step
K = I(1);
L = I(2);
disp(I');
end

% solution: K = 0.1972; L =  0.7887.

%% 4.4. Now instead suppose that the firm problem is
% max(K;L) pai(K,L) = 3K^(0.2)*L^(0.4)-e^(K+L)

%FOC on K = 0.2*3*K^(-0.8)*L^(0.4) - e^(K+L)
%FOC on L = 0.4*3*K^(0.2)*L^(-0.6) - e^(K+L)

I0 = [1;1];                           % Pick an initial guess
max4_4_handle = @(I) max4_4(I);       % Create the function handle
solution_opt = fsolve(max4_4_handle, I0)      % Use fsolve to solve the system

%solution_opt = 0.1636; 0.3271


%% 5. Now instead, linearize the first-order conditions you obtained in (4)
% around K0 = 1 and L0 = 2 and solve this linearized system using matrix
% inversion. Is the solution close to the one you found in (4)?

%No.

% unliner FOC:
%FOC on K = 0.2*3*K^(-0.8)*L^(0.4) - e^(K+L)
%FOC on L = 0.4*3*K^(0.2)*L^(-0.6) - e^(K+L)

%Tylar Extension:
%FOC1_taylor = (0.6 * 2^(0.4) - exp(3)) + (-0.48 * 2^(0.4) - exp(3)) * (K - 1) + (0.24 *↙
2^(-0.6) - exp(3)) * (L - 2);
%FOC2_taylor = (1.2 * 2^(-0.6) - exp(3)) + (0.24 * 2^(-0.6) - exp(3)) * (K - 1) + (-0.72↙
* 2^(-1.6) - exp(3)) * (L - 2);

%rearrange:
% (-0.48 * 2^(0.4) - exp(3)) * K + (0.24 * 2^(-0.6) - exp(3)) * L = -(1.08 * 2^(0.4) -↙
0.48 * 2^(-0.6) + 2 * exp(3))
% (0.24 * 2^(-0.6) - exp(3)) * K + (-0.72 * 2^(-1.6) - exp(3)) * L = -(0.96 * 2^(-0.6) +↙
1.44 * 2^(-1.6) + 2 * exp(3))

% rearrange to be metrix:
A = [-0.48 * 2^(0.4) - exp(3), 0.24 * 2^(-0.6) - exp(3); 0.24 * 2^(-0.6) - exp(3), -0.72↙
* 2^(-1.6) - exp(3)];
b = [-(1.08 * 2^(0.4) - 0.48 * 2^(-0.6) + 2 * exp(3)); -(0.96 * 2^(-0.6) + 1.44 * 2^↙
(-1.6) + 2 * exp(3))];

X = A\b;
K = X(1)
L = X(2)

% solution: K = 0.6815; L = 1.3630. No, they are not close.
```

```
function pai_deri = HW6_Cournot(q)
pai_deri = 1 - 2 * q - 2*q^(1/2)     % calculate the derivitive of pai
end
```

```
function pai_deri = HW6_Cournot2(q)
q1 = q(1)
q2 = q(2)
q3 = q(3)

pai1_deri = 1 - 2 * q1 - q2 - q3 - 2 * q1^(1/2);   % calculate the derivitive of pai1
pai2_deri = 1 - q1 - 2 * q2 - q3 - 2 * q1^(1/2);   % calculate the derivitive of pai2
pai3_deri = 1 - q1 - q2 - 2 * q3 - 2 * q1^(1/2);   % calculate the derivitive of pai3


pai_deri = [pai1_deri; pai2_deri; pai3_deri];     % return

end
```

```
function [f,g,H] = max4_2(I)
K = I(1);
L = I(2);

f = 3 * K^(0.2) * L^(0.4) - 2*K - L;
f = -f;                                                 % since max

% Supply the Gradient
%---------------------------------------------------------------------------
if nargout > 1                                          % nargout: Number of output arguments
    g = [0.6*K^(-0.8)*L^(0.4)-2;
         1.2*K^(0.2)*L^(-0.6)-1];
    g = -g;                                             % since max
end

% Supply the Hessian
%---------------------------------------------------------------------------
if nargout > 2
H = [-0.8*0.6*K^(-1.8)*L^0.4, 0.6*0.4*K^(-0.8)*L^(-0.6); 0.2*1.2*K^(-0.8)*L^(-0.6), -1.2↙
*0.6*K^0.2*L^(-1.6)];
H = -H;                                                 % since max
end

end
```

```
function F = max4_4(I)

K = I(1);                           % Define K as first element of I
L = I(2);                           % Define L as second element of I

F = [0.2*3*K^(-0.8)*L^(0.4) - exp(K+L); 0.4*3*K^(0.2)*L^(-0.6) - exp(K+L)]
                                    % Compute the function values for the given K,L
disp([K,L])
end
```