ECON90055 Computational Economics: Assignment 4

Zelin Chen - 797036

May 2, 2017

Problem 1

We want to pick the right θ to maximise the utility function, where θ is the amount spent on x, and $1-\theta$ is the amount spend on y. Given the utility maximisation problem: $max_\theta U(\theta)$ where $U(\theta) = (\frac{\theta}{2})^{1/2} + 2(\frac{1-\theta}{3})^{1/2}$. In order to find a maximum, we need to solve for its first derivative equals zero. Newton's Method suggests an iterative scheme:

$$x_{k+1} = x_k - \frac{U'(\theta)}{U''(\theta)}$$

where first and second derivative of $U(\theta)$ are:

$$U'(\theta) = \frac{1}{4}(\frac{\theta}{2})^{-\frac{1}{2}} - \frac{1}{3}(\frac{1-\theta}{3})^{-\frac{1}{2}}$$

$$U''(\theta) = -\frac{1}{16}(\frac{\theta}{2})^{-\frac{3}{2}} - \frac{1}{18}(\frac{1-\theta}{3})^{-\frac{3}{2}}$$

The result shows that under Newton's method, our iteration converges very quickly only in three periods. See from the table: [1]

Its code explains the function for calculating first derivative, second derivative and its main action screen are also included below the output table 1.

Listing 1: Code for Q1 Main Calculation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Computational Economics
% Zelin Chen 797036
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main for Q1
% NEWTON's METHOD
```

[1] I first used the stopping rule of 0.0001, it converges very quickly. So then I tried to let the iteration run for 20 terms, but it converges to the solution already in only 4 iterations. Therefore, I decide to only include 10 periods to display this iteration.

Table 1: Iterations of θ and errors

round	θ	errors
1	0.5000	-
2	0.2596	-0.2404
3	0.2724	0.0128
4	0.2727	0.0003
5	0.2727	0.0000
6	0.2727	0.0000
7	0.2727	0.0000
8	0.2727	0.0000
9	0.2727	0.0000
10	0.2727	0.0000

```
% here we have maximal problem, which we want to find f'(x)=0. so
% we write function as: x' = x - f'(x)/f''(x)

n=10;
% space to save result for theta and errors
theta_store = zeros(n,1);
err_store = zeros(n,1);
theta_store(1) = 0.5;
err_store(1) = 0.5;

term = 0;

while term < n %abs(err_store(term+1)) > 0.001
    term = term + 1;
    d2f = d2fx(theta_store(term)); % 2nd derivative
    df = dfx(theta_store(term));   % 1st derivative
    % Newton's Method - iteration
    theta_store(term+1) = theta_store(term) - df/d2f;
    err_store(term+1) = theta_store(term+1) - theta_store(term);
end


% produce Latex Table:
table.data = err_store;
table.makeCompleteLatexDocument = 1;
table.dataNanString = 'NaN';
table.tableColumnAlignment = 'c';
table.dataFormat = {'%.4f'};
latex = latexTable(table);
```

Listing 2: Code for Finding 1st Derivatives

```
function d = dfx(x)
    d = 0.25*(x/2)^(-0.5) - (1/3)*(((1-x)/3)^(-0.5));
end
```

Listing 3: Code for Finding 2nd Derivatives

```
function d = d2fx(x)
    d = (-1/16)*(x/2)^(-1.5) - (1/18)*(((1-x)/3)
        ^(-1.5));
end
```

Problem 2

To construct the profit function, we need to first derive the price function of Y and Z.

$$u_Y(Y,Z) = \eta Y^{\alpha-1}(Y^\alpha + Z^\alpha)^{\frac{\eta}{\alpha}-1}$$
$$u_Z(Y,Z) = \eta Z^{\alpha-1}(Y^\alpha + Z^\alpha)^{\frac{\eta}{\alpha}-1}$$

Profit function $\Pi(Y,Z)$ then will be:

$$\eta Y^\alpha(Y^\alpha + Z^\alpha)^{\frac{\eta}{\alpha}-1} + \eta Z^\alpha(Y^\alpha + Z^\alpha)^{\frac{\eta}{\alpha}-1} - 0.62Y - 0.60Z$$

It can be further simplified as:

$$\eta(Y^\alpha + Z^\alpha)^{\frac{\eta}{\alpha}} - 0.62Y - 0.60Z$$

Now substitute $Y = e^y$ and $Z = e^z$:

$$\Pi(e^y, e^z) = \pi(y,z) = \eta(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}} - 0.62e^y - 0.60e^z$$

Find first difference of y and z:

$$\pi_y = \frac{\partial \pi}{\partial y} = \eta^2 e^{\alpha y}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-1} - 0.62e^y$$

$$\pi_z = \frac{\partial \pi}{\partial z} = \eta^2 e^{\alpha z}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-1} - 0.60e^z$$

Also solve for second derivatives to form Hessian matrix

$$\pi_{yy} = \frac{\partial^2 \pi}{\partial y^2} = \alpha\eta^2(\frac{\eta}{\alpha} - 1)e^{2\alpha y}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-2} + \alpha\eta^2 e^{\alpha y}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-1} - 0.62e^y$$

$$\pi_{zz} = \frac{\partial^2 \pi}{\partial z^2} = \alpha\eta^2(\frac{\eta}{\alpha} - 1)e^{2\alpha z}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-2} + \alpha\eta^2 e^{\alpha z}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-1} - 0.60e^z$$

$$\pi_{yz} = \frac{\partial^2 \pi}{\partial y \partial z} = \frac{\partial^2 \pi}{\partial z \partial y} = \alpha\eta^2(\frac{\eta}{\alpha} - 1)e^{\alpha z + \alpha y}(e^{\alpha y} + e^{\alpha z})^{\frac{\eta}{\alpha}-2}$$

So now we can form the first derivative vector and second derivative matrix of Newton's methods:

$$\nabla\pi(y,z) = \begin{bmatrix}\pi_y\\ \pi_z\end{bmatrix}$$

$$\nabla^2\pi(y,z) = \begin{bmatrix}\pi_{yy} & \pi_{yz}\\ \pi_{zy} & \pi_{zz}\end{bmatrix}$$

The iteration scheme for Newton's Method is then:

$$\begin{bmatrix}y'\\ z'\end{bmatrix} = \begin{bmatrix}y\\ z\end{bmatrix} - \frac{\nabla\pi(y,z)}{\nabla^2\pi(y,z)}$$

For Broyden-Fletcher-Goldfarb-Shanno (BFGS) Method, the iteration scheme does not require for derivation of second derivatives, instead, it iterates the Hessian Matrix by using the difference in first derivatives and solution errors. Define change in first derivative as:

$$\boldsymbol{d} = \begin{bmatrix}\pi_y' - \pi_y\\ \pi_z' - \pi_z\end{bmatrix}$$

$$\boldsymbol{s} = \begin{bmatrix}y' - y\\ z' - z\end{bmatrix}$$

Then $\nabla^2\pi(y,z)$ in Newton's Method is replaced by $\boldsymbol{H}$ with iterative scheme:

$$\boldsymbol{H'} = \boldsymbol{H} - \frac{\boldsymbol{H}\boldsymbol{s}\boldsymbol{s}^{\boldsymbol{T}}\boldsymbol{H}}{\boldsymbol{s}^{\boldsymbol{T}}\boldsymbol{H}\boldsymbol{s}} + \frac{\boldsymbol{d}\boldsymbol{d}^{\boldsymbol{T}}}{\boldsymbol{d}^{\boldsymbol{T}}\boldsymbol{s}}$$

With this BFGS Method we can also approach the solution, but with different round of iterations. The following tables shows the difference in convergence between two methods. We see that Newton's Method actually converges to the solution quicker than BFGS Method. It takes Newton's Method 7 iterations to converge, on the contrast, BFGS spends 18 terms to reach the same answer.

Table 2: Iterations under Newton's and BFGS Methods

iteration	Newton's Method				BFGS Method			
	y	$changey$	z	$changez$	y	$changey$	z	$changez$
1	1.50	-0.61	2.00	-0.56	1.50	-0.61	0.00	-0.56
2	0.89	-0.48	1.44	-0.37	1.76	-0.63	0.02	-0.05
3	0.41	-0.36	1.07	-0.11	1.13	-0.18	-0.03	-0.03
4	0.05	-0.33	0.96	0.07	0.96	-0.16	-0.06	-0.05
5	-0.28	-0.20	1.04	0.03	0.79	-0.04	-0.11	-0.03
6	-0.49	-0.07	1.07	0.01	0.76	0.05	-0.14	0.06
7	-0.56	-0.01	1.08	0.00	0.81	0.01	-0.08	0.03
8	-0.56	-0.00	1.08	0.00	0.82	0.03	-0.05	0.20
9	-0.56	-0.00	1.08	0.00	0.85	-0.02	0.15	0.28
10	-0.56	0.00	1.08	-0.00	0.83	-0.12	0.43	0.27
11	-0.56	-0.00	1.08	0.00	0.71	-0.29	0.70	0.18
12	-0.56	0.00	1.08	-0.00	0.42	-0.29	0.88	-0.02
13	-0.56	-0.00	1.08	0.00	0.13	-0.32	0.85	0.20
14	-0.56	0.00	1.08	-0.00	-0.19	-0.12	1.06	-0.01
15	-0.56	-0.00	1.08	0.00	-0.31	-0.16	1.04	0.03
16	-0.56	0.00	1.08	-0.00	-0.48	-0.06	1.07	0.01
17	-0.56	-0.00	1.08	0.00	-0.54	-0.02	1.07	0.00
18	-0.56	0.00	1.08	-0.00	-0.56	-0.00	1.08	-0.00
19	-0.56	-0.00	1.08	0.00	-0.56	-0.00	1.08	0.00
20	-0.56	0.00	1.08	-0.00	-0.56	0.00	1.08	-0.00
21	-0.56	-0.00	1.08	0.00	-0.56	-0.00	1.08	0.00
22	-0.56	0.00	1.08	-0.00	-0.56	0.00	1.08	0.00
23	-0.56	-0.00	1.08	0.00	-0.56	0.00	1.08	-0.00
24	-0.56	0.00	1.08	-0.00	-0.56	-0.00	1.08	0.00
25	-0.56	0.00	1.08	0.00	-0.56	0.00	1.08	0.00

The following codes includes the main script that runs Newton's Method and BFGS Method to solve the solution, also two first derivative, three second derivative functions, and function that conducts updates on Hessian for BFGS Method.

Listing 4: Main Script for Newton's Method and BFGS Method Computation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computational Economics
% Zelin Chen 797036
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
clear all;
%-----------Newton's method ---------------------
a = 0.98;
n = 0.85;

size = 25;
% store values of x
yz = zeros(size,2);
yz(1,1) = 1.5;
yz(1,2) = 2.0;

err = zeros(size,2);
err(1,1) = 1.5;
err(1,2) = 2.0;

term = 0;

% save 1st and 2nd derivatives
J = zeros(2,2);
df = zeros(2,1);

while term < size-1

    term = term+1;
    % obtaining values for 1st and 2nd derivatives
    dy1 = dy(yz(term,1),yz(term,2),a,n);
    dyy1 = dyy(yz(term,1),yz(term,2),a,n);
    dz1 = dz(yz(term,1),yz(term,2),a,n);
    dzz1 = dzz(yz(term,1),yz(term,2),a,n);
    dyz1 = dzy(yz(term,1),yz(term,2),a,n);
    % Hessian Matrix
    J(1,1) = dyy1;
    J(1,2) = dyz1;
    J(2,1) = dyz1;
    J(2,2) = dzz1;
```

```
    invJ = (1/det(J)) * [J(2,2) -J(1,2); -J(2,1) J(1,1)];
    % 1st derivative vector
    df(1,1) = dy1;
    df(2,1) = dz1;

    % iterative scheme for y',z'
    s = invJ * -df;
    yz(term+1,1) = yz(term,1) + s(1);
    yz(term+1,2) = yz(term,2) + s(2);
    % store steps
    err(term,:) = s;

end
% store Newton's Method iterations
Solution = zeros(size,8);
Solution(:,1) = yz(:,1);
Solution(:,2) = err(:,1);
Solution(:,3) = yz(:,2);
Solution(:,4) = err(:,2);

%-----------BFGS method ---------------------
% store values of x
yz = zeros(size,2);
yz(1,1) = 1.5; %y
yz(2,1) = 2.0; %z

% save Hessian
H = zeros(2,2);

% first iteration, 1st guess of Hessian is Identity matrix
H_tem = [1 0; 0 1];

% solve for first update of 1st derivative
dy1 = dy(yz(1,1),yz(1,2),a,n);
dz1 = dz(yz(1,1),yz(1,2),a,n);
% 1st derivative vector
df = zeros(2,1);
df(1) = dy1;
df(2) = dz1;
df_tem = df;

invH = (1/det(H_tem)) * [H_tem(2,2) -H_tem(1,2); -H_tem(2,1) H_tem(1,1)];
% solve for s:step
```

```
s = invH * -df;
%s_tem = s;
yz(2,1) = yz(1,1) + s(1);
yz(2,2) = yz(1,2) + s(2);
term = 1;

while term < size-1
    term = term + 1;

    % update Hessian
    dy1 = dy(yz(term,1),yz(term,2),a,n);
    dz1 = dz(yz(term,1),yz(term,2),a,n);
    df(1) = dy1;
    df(2) = dz1;
    ddf = df - df_tem;% change of 1st derivatives:

    % BFGS function takes three inputs, H,s,d to produce updated H'
    H = BFGS(H_tem,s,ddf);

    % with new H, start updating y' and z'
    invH = (1/det(H)) * [H(2,2) -H(1,2); -H(2,1) H(1,1)];
    s = invH * -df;
    yz(term+1,1) = yz(term,1) + s(1);
    yz(term+1,2) = yz(term,2) + s(2);

    % store temporary values
    df_tem =df;
    H_tem = H;
    err(term,:) = s;
end

Solution(:,5) = yz(:,1);
Solution(:,6) = err(:,1);
Solution(:,7) = yz(:,2);
Solution(:,8) = err(:,2);

%-----------------Latex table-----------------------
% combine value and errors
table.data = Solution;
table.tableColLabels = {'$y$','$change y$','$z$','$change z$','$y$','$change y$','$z$','$change z$'};
table.tableRowLabels ={'1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25'};
```

```
table.dataFormat = {'%.2f'};

table.makeCompleteLatexDocument = 1;

% generate LaTex code
latex = latexTable(table);
```

Listing 5: Function for First Derivative on y

```
function val = dy(y,z,a,n)
    na1   = (n/a) - 1;
    na2   = (n/a) - 2;
    u_yz = exp(a*y)+exp(a*z);

    val =  n^2 * exp(a*y) * (u_yz)^(na1) - 0.62*exp(y);
end
```

Listing 6: Function for First Derivative on z

```
function val = dz(y,z,a,n)
    na1   = (n/a) - 1;
    na2   = (n/a) - 2;
    u_yz = exp(a*y)+exp(a*z);

    val = n^2 * exp(a*z) * (u_yz)^(na1) - 0.60*exp(z);
end
```

Listing 7: Function for Second Derivative on y,y

```
function val = dyy(y,z,a,n)
    u_yz = exp(a*y)+exp(a*z);
    na1   = (n/a) - 1;
    na2   = (n/a) - 2;

    val = a*n^2 * exp(a*y) * (u_yz)^(na1) + a * n^2 * (na1) * exp(2*a*y) * (u_yz)^(na2)- 0.62 * exp(y);
end
```

Listing 8: Function for Second Derivative on z,z

```
function val = dzz(y,z,a,n)

    u_yz = exp(a*y)+exp(a*z);
    na1   = (n/a) - 1;
    na2   = (n/a) - 2;
```

```
    val = a*n^2 * exp(a*z) * (u_yz)^(na1) + a * n^2 *
        (na1) * exp(2*a*z) * (u_yz)^(na2) - 0.6 * exp(z
        );

end
```

Listing 9: Function for Second Derivative on y,z

```
function val = dzy(y,z,a,n)
    u_yz = exp(a*y)+exp(a*z);
    na1   = (n/a) - 1;
    na2   = (n/a) - 2;

    val = a * n^2 * na1 * (u_yz)^na2 * exp(a*y+a*z);
end
```

Listing 10: Function for BFGS Hessian updates

```
function H = BFGS(H_tem,s,ddf)

    H = H_tem - (H_tem * s * transpose(s)* H_tem)/(
        transpose(s)*H_tem*s) + (ddf*transpose(ddf))/(
        transpose(ddf)*s);

end
```