Computational Economics: Problem Set 2

Yangming Bao, ID: 5601239
Cheung Ying Lun, ID: 5441897

Problem 1: Getting Started with Github

When we set `racism=0.5`, we get an error message. The error occurs because in line 50 and 51 the command `nrow_mover = length(mover)` and `nrow_freehouse = length(freehouse)` respectively. However, when there is only one or two movers, the `length` commands returns three since there are three columns. Therefore, one should use `size(mover,1)` and `size(freehouse,1)` instead to get the number of rows.

Problem 2: Univariate Problems

Write it as a univariate problem with the corresponding parameters,

$$0.5q + q^{0.5} - 2 = 0$$

Thus the root for the equation can be calculated analytically by letting $x = q^{0.5}$, then

$$\begin{aligned} & x^2 + 2x - 4 = 0 \\ \Rightarrow & x = -1 \pm \sqrt{5} \end{aligned}$$

Thus $q = x^2 = 6 \pm 2\sqrt{5}$.

When compute it with bisection algorithm, it gives the value $q = 1.5279$. When compute it with bisection algorithm, it gives the value $q = 1.5279$. When implementing a Gauss-Seidel fixed-point iteration, it converges with $(q, p) = (1.5279, 2.2361)$, but not converge with the other order. When dampening with $\lambda = 0.5$, it converges with the value $(p, q) = (2.2361, 1.5278)$.

Problem 3: A Contribution to the Empirics of Economic Growth

Dependent varibale: log difference GDP per working-age person 1960-1985			
Sample	Non-oil	Intermediate	OECD
Observation	98	75	22
Constant	1.874	2.498	4.155
	(0.828)	(0.860)	(0.898)
ln(Y60)	-0.288	-0.366	-0.398
	(0.060)	(0.066)	(0.063)
ln(I/GDP)	0.524	0.538	0.332
	(0.085)	(0.099)	(0.156)
ln(n+g+δ)	-0.506	-0.545	-0.863
	(0.283)	(0.281)	(0.304)
ln(School)	0.231	0.270	0.228
	(0.058)	(0.078)	(0.130)
$\bar{R}^2$	0.46	0.43	0.63

Problem 4: Solving The Augmented Solow Growth Model

1. The system of equations reads

$$Y(t) = K(t)^{\alpha_K} H(t)^{\alpha_H} [A(t)L(t)]^{1-\alpha_K-\alpha_H} \tag{1}$$

$$\dot{K}(t) = s_K Y(t) - \delta_k K(t) \tag{2}$$

$$\dot{H}(t) = s_H Y(t) - \delta_h H(t) \tag{3}$$

$$\dot{L}(t) = nL(t) \tag{4}$$

$$\dot{A}(t) = gA(t) \tag{5}$$

where $\dot{X}(t)$ denotes the time derivative of time function $X(t)$. Now transform the system and divide all variables by $A(t)L(t)$. We denote the variables after transformation by $x(t) = X(t)/[A(t)L(t)]$. We obtain after some steps

$$y(t) = k(t)^{\alpha_K} h(t)^{\alpha_H} \tag{6}$$
$$\dot{k}(t) = s_K y(t) - (n + g + \delta_k)k(t) \tag{7}$$
$$\dot{h}(t) = s_H y(t) - (n + g + \delta_h)h(t) \tag{8}$$

At the steady-state, $\dot{k} = \dot{h} = 0$, thus we have

$$s_K y(t) = (n + g + \delta_k)k(t) \tag{9}$$
$$s_H y(t) = (n + g + \delta_h)h(t) \tag{10}$$

Substituting y into the equations and solving for k^* and h^* we obtain

$$k^* = \left(\frac{s_K}{n + g + \delta_k}\right)^{\frac{1-\alpha_H}{1-\alpha_K-\alpha_H}} \left(\frac{s_H}{n + g + \delta_h}\right)^{\frac{\alpha_H}{1-\alpha_K-\alpha_H}} \tag{11}$$
$$h^* = \left(\frac{s_H}{n + g + \delta_h}\right)^{\frac{1-\alpha_K}{1-\alpha_K-\alpha_H}} \left(\frac{s_K}{n + g + \delta_k}\right)^{\frac{\alpha_K}{1-\alpha_K-\alpha_H}} \tag{12}$$

Now substituting in our parameterization, we obtain

$$k^* \approx 5.7932, \qquad h^* \approx 8.5194. \tag{13}$$

2. Substituting Eq.(6) into Eq.(7) and (8), and letting $\dot{k} = \dot{h} = 0$, we obtain at steady state

$$f_1(k, h) := s_K k(t)^{\alpha_K} h(t)^{\alpha_H} - (n + g + \delta_k)k(t) = 0 \tag{14}$$
$$f_2(k, h) := s_H k(t)^{\alpha_K} h(t)^{\alpha_H} - (n + g + \delta_h)h(t) = 0 \tag{15}$$

Dropping all time arguments, the Jacobian is given by

$$\mathcal{J} = \begin{pmatrix} \frac{\partial f_1}{\partial k} & \frac{\partial f_1}{\partial h} \\ \frac{\partial f_2}{\partial k} & \frac{\partial f_2}{\partial h} \end{pmatrix} \tag{16}$$
$$= \begin{pmatrix} \alpha_K s_K k^{\alpha_K - 1} h^{\alpha_H} - (n + g + \delta_k) & \alpha_H s_K k_K^{\alpha} h^{\alpha_H - 1} \\ \alpha_K s_H k^{\alpha_K - 1} h^{\alpha_H} & \alpha_H s_H k_K^{\alpha} h^{\alpha_H - 1} - (n + g + \delta_k) \end{pmatrix} \tag{17}$$

3. For $\epsilon = \delta = 0.001$, convergence is reported after 5 iterations, with infinite norm difference from the true value being 3.133e-11.

   For $\epsilon = \delta = 1e - 20$, convergence is reported after 7 iterations, with infinite norm difference from the true value being 7.105e-15.

4. For $\epsilon = \delta = 0.001$, convergence is reported after 21 iterations, with infinite norm difference from the true value being 0.00019.
   For $\epsilon = \delta = 1e-20$, convergence is reported after 29 iterations, with infinite norm difference from the true value being 1.776e-15.

5. For $\epsilon = \delta = 0.001$, convergence is reported after 21 iterations, with infinite norm difference from the true value being 0.00019.
   For $\epsilon = \delta = 1e-20$, convergence is reported after 29 iterations, with infinite norm difference from the true value being 1.776e-15.

6. Convergence of k and h are shown in the figure below. Agents start to accumulate both types of capital since the beginning, at a decreasing rate when approaching the steady state. They stop capital accumulation after reaching the steady state and continue to produce with the same amount of captial.

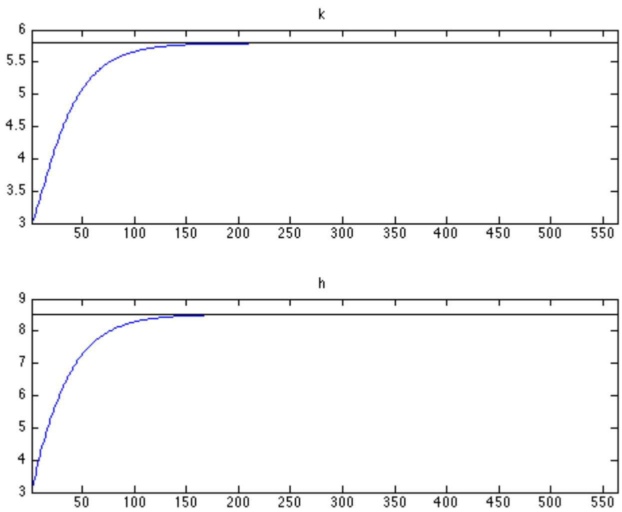

Figure 1: Convergence of k and h

Problem 5: Cournot Oligopoly

With $n = 2$, the equilibrium allocations is $(0.83957, 0.6888)$;

With $n = 5$, the equilibrium allocations is $(0.7213, 0.67382, 0.59542, 0.50695, 0.42312)$;

With $n = 10$, the equilibrium allocations is

$$(0.60977, 0.58986, 0.55369, 0.50706, 0.45587, 0.40478, 0.35681, 0.31346, 0.27524, 0.24205).$$

Main Code

Question 1

```
close all; clear all
clc;


%% Step 1: Initial Allocation of Households on the Checkboard
rand('seed',0);     % initializes the random number generator
parcel = randperm(225)';
resarea = zeros(15,15);
resarea(parcel(1:110)) = 1;        % black
resarea(parcel(111:220)) = -1;     % whites
clear parcel;

resareaaug = zeros(17,17); resareaaug(2:16,2:16) = resarea;
subplot(2,2,1);
imagesc(resareaaug); % create a colored plot of the matrix values
colormap(flipud(gray));
title(['initial distribution']);


%% Step 2: Dynamics
racism = 0.5;
numplot = 1;
for niter = 1:45;
    mover = [];      % the matrix mover includes all indices of people that
        are looking for a new house
    freehouse = []; % the matrix freehouse includes all indices of ex post
        free houses
    for i = 2:16;
        for j = 2:16
            neighbors = resareaaug( (i-1):(i+1), (j-1):(j+1) );
            DD_white = logical(neighbors == -1);
            DD_free = logical(neighbors == 0);
            DD_black = logical(neighbors == 1);

            if resareaaug(i,j) == -1        % identify white
                btow = (sum(DD_black(1,:)) + DD_black(2,1) + DD_black(2,3)
                    + sum(DD_black(3,:)))/(8-sum(sum(DD_free))); %(sum(
```

```
                    DD_white(1,:)) + DD_white(2,1) + DD_white(2,3) + sum(
                    DD_white(3,:)));
                if btow > racism
                    mover = [mover; i,j,-1];
                end
            elseif resareaaug(i,j) == 1        % identify black
                wtob = (sum(DD_white(1,:)) + DD_white(2,1) + DD_white(2,3)
                    + sum(DD_white(3,:)))/(8-sum(sum(DD_free))); %(sum(
                    DD_black(1,:)) + DD_black(2,1) + DD_black(2,3) + sum(
                    DD_black(3,:)));
                if wtob > racism
                    mover = [mover; i,j,1];
                end
            elseif resareaaug(i,j) == 0      % identify non-occupied houses
                freehouse = [freehouse; i,j,0];
            end
        end
    end
    freehouse = [freehouse; mover];      % free houses is previously free
        and just becoming free

    nrow_mover = size(mover,1);
    nrow_freehouse = size(freehouse,1);
    parcel = randperm(nrow_freehouse)';      % prepare random allocation of
        movers to free houses by randomizing houses
    for k = 1:nrow_freehouse;
        if k <= nrow_mover
            resareaaug(freehouse(parcel(k),1),freehouse(parcel(k),2)) =
                mover(k,3);
        else
            resareaaug(freehouse(parcel(k),1),freehouse(parcel(k),2)) = 0;
        end
    end
    clear parcel;

    if rem(niter,15) == 0
        numplot = numplot+1;
        subplot(2,2,numplot);
        imagesc(resareaaug);
        colormap(flipud(gray));
        title(['iteration ', num2str(niter)]);
```

```
    end
    %fprintf('iter = %.3d\n',niter);
end
```

Question 2

```

%% 2.2
clc;clear;

f1=@(x)x^3+4-1/x;
[x1, ~] = bisection(f1,[0.1;1],[10^-6;10^-6]);
disp(['the root is' ' ' num2str(x1)]);

f2=@(x)-exp(-x)+exp(-x^2);
[x2, ~] = bisection(f2,[0.2;2],[10^-6;10^-6]);
disp(['the root is' ' ' num2str(x2)]);


%% 2.3
f = @(q)q^0.5+0.5*q-2;

% with bisection
[q, ~] = bisection(f,[1;2],[10^-6;10^-6]);
disp(['the root is' ' ' num2str(q)]);
% with fzero
q = fzero(f,1);
disp(['the root is' ' ' num2str(q)]);
% Gauss seidel fixed point
a = 3;
b = 0.5;
c = 1;
d = 1;
psi = 0.5;

%X=[q,p]

g1 = @(X)X(2)-a+b*X(1);
g2 = @(X)X(2)-c-d*X(1)^psi;
dg1 = @(X)b;
dg2 = @(X)1;
```

```

g = {g1,g2};
dg = {dg1,dg2};

ini_val = [0.1;0.1];
eps = 0.00001;
del = 0.001;
max_it = 10000;
dampening = 1;
stop_crit = [eps,del,max_it];

X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);

%X=[p,q]
g1 = @(X)X(1)-a+b*X(2);
g2 = @(X)X(1)-c-d*X(2)^psi;
dg1 = @(X) 1;
dg2 = @(X)-d*psi*X(2)^(psi-1);

g = {g1,g2};
dg = {dg1,dg2};
X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);

dampening = 0.5;
X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);
```

Question 3

```
clc;clear

%% 3.1
% load data
[data txt]= xlsread('MRW92QJE-data.xls');
data(:,2) = [];

% delete countries with missing data
No_Countries = size(data,1);
index = [];
for iCountry = 1:No_Countries
    if sum(isnan(data(iCountry,:)))>0
        index=[index iCountry];
```

```
    end
end
data(index,:)=[];

%% 3.2
index_nonoil = [];
index_inter = [];
index_oecd = [];

for iCountry = 1:length(data)
    % non-oil countries
    if data(iCountry,2)==1
        index_nonoil = [index_nonoil iCountry];
    end
    % intermediate countries
    if data(iCountry,3)==1
        index_inter = [index_inter iCountry];
    end
    % oecd countries
    if data(iCountry,4)==1
        index_oecd = [index_oecd iCountry];
    end

end

nonoil_country = data(index_nonoil,:);
inter_country = data(index_inter,:);
oecd_country = data(index_oecd,:);

%% 3.3
%nonoil
gdp60_nonoil = log(nonoil_country(:,5));
gdp85_nonoil = log(nonoil_country(:,6));
iy_nonoil = log(nonoil_country(:,9));
school_nonoil = log(nonoil_country(:,10));

y_nonoil = gdp85_nonoil-gdp60_nonoil;
X_nonoil = [ones(size(y_nonoil,1),1) gdp60_nonoil iy_nonoil ...
     log(nonoil_country(:,8)+5) school_nonoil];
Beta_nonoil = (X_nonoil'*X_nonoil)\(X_nonoil'*y_nonoil);
sigma_nonoil = var(y_nonoil-X_nonoil*Beta_nonoil)*...
```

```
    (X_nonoil'*X_nonoil/size(y_nonoil,1))^(-1)./size(y_nonoil,1);

%intermediate country
gdp60_inter = log(inter_country(:,5));
gdp85_inter = log(inter_country(:,6));
iy_inter = log(inter_country(:,9));
school_inter = log(inter_country(:,10));

y_inter = gdp85_inter-gdp60_inter;
X_inter = [ones(size(y_inter,1),1) gdp60_inter iy_inter ...
    log(inter_country(:,8)+5) school_inter];
Beta_inter = (X_inter'*X_inter)\(X_inter'*y_inter);
sigma_inter = var(y_inter-X_inter*Beta_inter)*...
    (X_inter'*X_inter/size(y_inter,1))^(-1)./size(y_inter,1);

%oecd country
gdp60_oecd = log(oecd_country(:,5));
gdp85_oecd = log(oecd_country(:,6));
iy_oecd = log(oecd_country(:,9));
school_oecd = log(oecd_country(:,10));

y_oecd = gdp85_oecd-gdp60_oecd;
X_oecd = [ones(size(y_oecd,1),1) gdp60_oecd iy_oecd ...
    log(oecd_country(:,8)+5) school_oecd];
Beta_oecd = (X_oecd'*X_oecd)\(X_oecd'*y_oecd);
sigma_oecd = var(y_oecd-X_oecd*Beta_oecd)*...
    (X_oecd'*X_oecd/size(y_oecd,1))^(-1)./size(y_oecd,1);

%% 3.4

R2_nonoil = 1 -    var(y_nonoil-mean(y_nonoil))\var(y_nonoil-X_nonoil*
    Beta_nonoil)...
    *((length(y_nonoil)-1)/(length(y_nonoil)-1-length(Beta_nonoil)));
R2_inter = 1 -    var(y_inter-mean(y_inter))\var(y_inter-X_inter*Beta_inter)
    ...
    *((length(y_inter)-1)/(length(y_inter)-1-length(Beta_inter)));
R2_oecd = 1 -    var(y_oecd-mean(y_oecd))\var(y_oecd-X_oecd*Beta_oecd)...
    *((length(y_oecd)-1)/(length(y_oecd)-1-length(Beta_oecd)));


clearvars -except Beta* R2* sigma*
```

```
Beta = [Beta_nonoil Beta_inter Beta_oecd];
Std_Beta = [(diag(sigma_nonoil)).^0.5 (diag(sigma_inter)).^0.5 (diag(
    sigma_oecd)).^0.5];
R2 = [R2_nonoil R2_inter R2_oecd];
```

Question 4

```
% Computational Economics PS2 Q4

clear, clc
close all

%% Parameter and function initialization
alpha_k = 0.33;
alpha_h = 0.33;
s_k = 0.2;
s_h = 0.2;
delta_k = 0.1;
delta_h = 0.06;
n = 0.01;
g = 0.015;
ini_val = [3,3];
ini_Jac = eye(2);

fun = @(val) steady_state(val,alpha_k,alpha_h,s_k,s_h,...
                        delta_k,delta_h,n,g);
Jac = @(val) Jacobian(val,alpha_k,alpha_h,s_k,s_h,...
                        delta_k,delta_h,n,g);
fun_fp = @(val) steady_state_fixed_pt(val,alpha_k,alpha_h,s_k,s_h,...
                        delta_k,delta_h,n,g);

%% Q4.1
k_star = (s_k/(n+g+delta_k))^((1-alpha_h)/(1-alpha_k-alpha_h))*...
            (s_h/(n+g+delta_h))^((alpha_h)/(1-alpha_k-alpha_h));
h_star = (s_h/(n+g+delta_h))^((1-alpha_k)/(1-alpha_k-alpha_h))*...
            (s_k/(n+g+delta_k))^((alpha_k)/(1-alpha_k-alpha_h));
x_star = [k_star;h_star];

%% Q4.3
eps = 0.001;
```

```
del = 0.001;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Newton_Method(fun,Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

eps = 1.e-20;
del = 1.e-20;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Newton_Method(fun,Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

%% Q4.4
eps = 0.001;
del = 0.001;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Broyden_Method(fun,ini_Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

eps = 1.e-20;
del = 1.e-20;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Broyden_Method(fun,ini_Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

%% Q4.5
eps = 0.001;
del = 0.001;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Inverse_Broyden_Method(fun,ini_Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

```

```
eps = 1.e-20;
del = 1.e-20;
max_it = 1000;
stop_crit = [eps,del,max_it];

x_hat = Inverse_Broyden_Method(fun,ini_Jac,ini_val,stop_crit);
disp(['Inf norm = ',num2str(max(abs(x_hat-x_star)))])

%% Q4.6
eps = 1.e-10;
del = 0.001;
max_it = 10000;
stop_crit = [eps,del,max_it];

[x_hat,x_history] = Fixed_Point_Method(fun_fp,ini_val,stop_crit,true);

figure
subplot(2,1,1)
plot(1:size(x_history,2),x_history(1,:))
hold on
plot([1,size(x_history,2)],[k_star,k_star],'k')
xlim([1,size(x_history,2)])
title('k')

subplot(2,1,2)
plot(1:size(x_history,2),x_history(2,:))
hold on
plot([1,size(x_history,2)],[h_star,h_star],'k')
xlim([1,size(x_history,2)])
title('h')
```

Question 5

```
function Q5
clc; clear;

EqulibriumFunc = @(q)func(q,1.6);

n=[2;5;10];

for i =1:length(n)
```

```

ini_Jac=eye(n(i));
ini_val = ones(n(i),1);

eps = 1.e-10;
del = 1.e-10;
max_it = 10000;
stop_crit = [eps,del,max_it];

q = Broyden_Method(EqulibriumFunc,ini_Jac,ini_val,stop_crit);
disp(['Equilibrium Output q = ',num2str(q')]);
end
end

function f = func( q,lambda )
% This function gives the equilibrium output levels.

n=length(q);
xi = 0.6;
f = zeros(n,1);

for i = 1:n
    xi = xi+(i-1)*0.2/(n-1);
    f(i) = sum(q)^(-1/lambda)-1/lambda*sum(q)^(-1/lambda-1)*q(i)-xi*q(i);
end

end
```

Functions

Question 2

```
function [x fx] = bisection(f,x,cc )
% This function calculate the root for f using bisection method.

xl = x(1,1);
xh = x(2,1);
fl = f(xl);
fh = f(xh);

```

```
tole = cc(1,1);
told = cc(2,1);

if fl*fh>0
    disp('initial [xl, xh] don''t bracket a root');
    x = -Inf; fx = -Inf; ef = 0;
    return
end

% bisection
while (xh-xl)>tole*(1+abs(xl)+abs(xh)) || abs(fm)>told
    xm = (xl+xh)/2;
    fm = f(xm);
    if fl*fm < 0
        xh = xm; fh = fm;
    else
        xl = xm; fl = fm;
    end

end
x = xm; fx = fm;
end
```

Question 4

```
function roots = Broyden_Method(func,ini_Jac,...
                                ini_val,...
                                stop_crit)
%This function perform the Newton's method for root-finding problem.
%         func: a function handle for value of the root-finding problem.
%          Jac: a function handle of the Jacobian function of the problem
%      ini_val: initial value
%    stop_crit: stopping criteria = [eps,del,max_it]

eps = stop_crit(1);
del = stop_crit(2);
max_it = stop_crit(3);
it = 0;
cont = true;

if length(ini_val)==size(ini_val,2)
```

```
    xold = ini_val';
else
    xold = ini_val;
end

J = ini_Jac;
fold = func(xold);

while cont
    it = it+1;

    xnew = xold-J\fold;
    fnew = func(xnew);
    dx = xnew-xold;
    J  = J+((fnew-fold-J*(dx))*dx')/(dx'*dx);
    if (norm(xold-xnew)<=eps*(1+norm(xnew))) || (it==max_it)
        cont = false;
    end
    xold = xnew;
    fold = fnew;
end

if norm(func(xnew))<=del
    disp(['Convergence after ',num2str(it),' iterations.'])
    roots = xnew;
else
    disp('Convergence failed.')
    roots = [];
end

end
```

```
function roots = Inverse_Broyden_Method(func,ini_Jac,...
                                        ini_val,...
                                        stop_crit)
%This function perform the Newton's method for root-finding problem.
%        func: a function handle for value of the root-finding problem.
%         Jac: a function handle of the Jacobian function of the problem
%     ini_val: initial value
%   stop_crit: stopping criteria = [eps,del,max_it]

eps = stop_crit(1);
```

```
del = stop_crit(2);
max_it = stop_crit(3);
it = 0;
cont = true;

if length(ini_val)==size(ini_val,2)
    xold = ini_val';
else
    xold = ini_val;
end

B = inv(ini_Jac);
fold = func(xold);

while cont
    it = it+1;

    xnew = xold-B*fold;
    fnew = func(xnew);
    dx = xnew-xold;
    df = fnew-fold;
    B  = B+((dx-B*df)*dx'*B)/(dx'*B*df);
    if (norm(xold-xnew)<=eps*(1+norm(xnew))) || (it==max_it)
        cont = false;
    end
    xold = xnew;
    fold = fnew;
end

if norm(func(xnew))<=del
    disp(['Convergence after ',num2str(it),' iterations.'])
    roots = xnew;
else
    disp('Convergence failed.')
    roots = [];
end

end
```

```
function J = Jacobian(val,alpha_k,alpha_h,s_k,s_h,...
                        delta_k,delta_h,n,g)
%This function calculate the Jacobian of the human capital augmented Solow
```

```
%growth model.

k = val(1);
h = val(2);

J = ...
    [alpha_k*s_k*k^(alpha_k-1)*h^alpha_h-(n+g+delta_k),...
     alpha_h*s_k*k^alpha_h*h^(alpha_h-1);...
     alpha_k*s_h*k^(alpha_k-1)*h^alpha_h,...
     alpha_h*s_h*k^alpha_h*h^(alpha_h-1)-(n+g+delta_h)];

end
```

```
function roots = Newton_Method(func,Jac,...
                               ini_val,...
                               stop_crit)
%This function perform the Newton's method for root-finding problem.
%       func: a function handle for value of the root-finding problem.
%        Jac: a function handle of the Jacobian function of the problem
%    ini_val: initial value
%  stop_crit: stopping criteria = [eps,del,max_it]

eps = stop_crit(1);
del = stop_crit(2);
max_it = stop_crit(3);
it = 0;
cont = true;

if length(ini_val)==size(ini_val,2)
    xold = ini_val';
else
    xold = ini_val;
end

while cont
    it = it+1;
    xnew = xold-Jac(xold)\func(xold);
    if (norm(xold-xnew)<=eps*(1+norm(xnew))) || (it==max_it)
        cont = false;
    end
    xold = xnew;
end
```

```

if norm(func(xnew))<=del
    disp(['Convergence after ',num2str(it),' iterations.'])
    roots = xnew;
else
    disp('Convergence failed.')
    roots = [];
end

end
```

```
function f = steady_state_fixed_pt(val,alpha_k,alpha_h,s_k,s_h,...
                          delta_k,delta_h,n,g)
%This function compute the value of the functions characterizing the steadt
%state of the human capital augmented Solow grotwh model.

k = val(1);
h = val(2);
y = k^alpha_k*h^alpha_h;

f = [...
    s_k*y-(n+g+delta_k -1)*k;...
    s_h*y-(n+g+delta_h -1)*h];

end
```

```
function f = steady_state(val,alpha_k,alpha_h,s_k,s_h,...
                          delta_k,delta_h,n,g)
%This function compute the value of the functions characterizing the steadt
%state of the human capital augmented Solow grotwh model.

k = val(1);
h = val(2);
y = k^alpha_k*h^alpha_h;

f = [...
    s_k*y-(n+g+delta_k)*k;...
    s_h*y-(n+g+delta_h)*h];

end
```