

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

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

$$\dot{K}(t) = s_K Y(t) - \delta_K K(t) \quad (2)$$

$$\dot{H}(t) = s_H Y(t) - \delta_H H(t) \quad (3)$$

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

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

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

$$y(t) = k(t)^{\alpha_K} h(t)^{\alpha_H} \quad (6)$$

$$\dot{k}(t) = s_K y(t) - (n + g + \delta_k)k(t) \quad (7)$$

$$\dot{h}(t) = s_H y(t) - (n + g + \delta_h)h(t) \quad (8)$$

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

$$s_K y(t) = (n + g + \delta_k)k(t) \quad (9)$$

$$s_H y(t) = (n + g + \delta_h)h(t) \quad (10)$$

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

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

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

Now substituting in our parameterization, we obtain

$$k^* \approx 5.7932, \quad h^* \approx 8.5194. \quad (13)$$

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

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

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

Dropping all time arguments, the Jacobian is given by

$$\mathcal{J} = \begin{pmatrix} \frac{\partial f_1}{\partial k} & \frac{\partial f_1}{\partial h} \\ \frac{\partial f_2}{\partial k} & \frac{\partial f_2}{\partial h} \end{pmatrix} \quad (16)$$

$$= \begin{pmatrix} \alpha_K s_K k^{\alpha_K-1} h^{\alpha_H} - (n + g + \delta_k) & \alpha_H s_K k^{\alpha_K} h^{\alpha_H-1} \\ \alpha_K s_H k^{\alpha_K-1} h^{\alpha_H} & \alpha_H s_H k^{\alpha_K} h^{\alpha_H-1} - (n + g + \delta_h) \end{pmatrix} \quad (17)$$

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 capital.

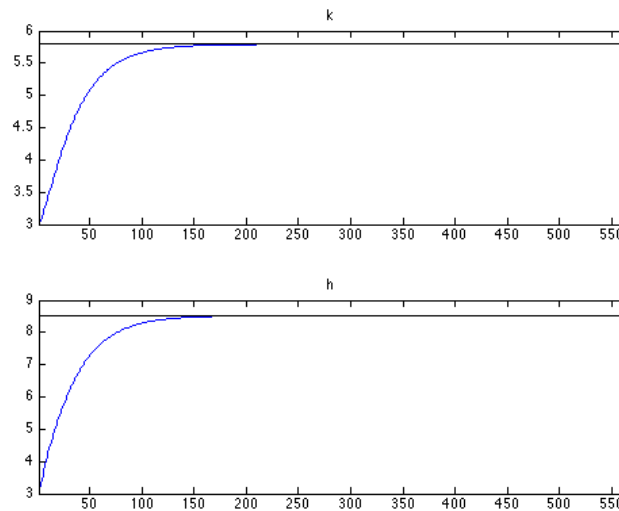


Figure 1: Convergence of k and h

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

```

1  close all; clear all
2  clc;
3
4
5  %% Step 1: Initial Allocation of Households on the Checkboard
6  rand('seed',0);      % initializes the random number generator
7  parcel = randperm(225)';
8  resarea = zeros(15,15);
9  resarea(parcel(1:110)) = 1;      % black
10 resarea(parcel(111:220)) = -1;   % whites
11 clear parcel;
12
13 resareaaug = zeros(17,17); resareaaug(2:16,2:16) = resarea;
14 subplot(2,2,1);
15 imagesc(resareaaug); % create a colored plot of the matrix values
16 colormap(flipud(gray));
17 title(['initial distribution']);
18
19
20 %% Step 2: Dynamics
21 racism = 0.5;
22 numplot = 1;
23 for niter = 1:45;
24     mover = []; % the matrix mover includes all indices of people that
                % are looking for a new house
25     freehouse = []; % the matrix freehouse includes all indices of ex post
                % free houses
26     for i = 2:16;
27         for j = 2:16
28             neighbors = resareaaug( (i-1):(i+1), (j-1):(j+1) );
29             DD_white = logical(neighbors == -1);
30             DD_free = logical(neighbors == 0);
31             DD_black = logical(neighbors == 1);
32
33             if resareaaug(i,j) == -1      % identify white
34                 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,:));
35     if btow > racism
36         mover = [mover; i,j,-1];
37     end
38     elseif resareaaug(i,j) == 1           % identify black
39         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,:));
40     if wtob > racism
41         mover = [mover; i,j,1];
42     end
43     elseif resareaaug(i,j) == 0           % identify non-occupied houses
44         freehouse = [freehouse; i,j,0];
45     end
46 end
47 end
48 freehouse = [freehouse; mover];           % free houses is previously free
        and just becoming free
49
50 nrow_mover = size(mover,1);
51 nrow_freehouse = size(freehouse,1);
52 parcel = randperm(nrow_freehouse)';       % prepare random allocation of
        movers to free houses by randomizing houses
53 for k = 1:nrow_freehouse;
54     if k <= nrow_mover
55         resareaaug(freehouse(parcel(k),1),freehouse(parcel(k),2)) =
            mover(k,3);
56     else
57         resareaaug(freehouse(parcel(k),1),freehouse(parcel(k),2)) = 0;
58     end
59 end
60 clear parcel;
61
62 if rem(niter,15) == 0
63     numplot = numplot+1;
64     subplot(2,2,numplot);
65     imagesc(resareaaug);
66     colormap(flipud(gray));
67     title(['iteration ', num2str(niter)]);

```

```

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

```

Question 2

```

1
2 %% 2.2
3 clc;clear;
4
5 f1=@(x)x^3+4-1/x;
6 [x1, ~] = bisection(f1,[0.1;1],[10^-6;10^-6]);
7 disp(['the root is' ' ' num2str(x1)]);
8
9 f2=@(x)-exp(-x)+exp(-x^2);
10 [x2, ~] = bisection(f2,[0.2;2],[10^-6;10^-6]);
11 disp(['the root is' ' ' num2str(x2)]);
12
13
14 %% 2.3
15 f = @(q)q^0.5+0.5*q-2;
16
17 % with bisection
18 [q, ~] = bisection(f,[1;2],[10^-6;10^-6]);
19 disp(['the root is' ' ' num2str(q)]);
20 % with fzero
21 q = fzero(f,1);
22 disp(['the root is' ' ' num2str(q)]);
23 % Gauss seidel fixed point
24 a = 3;
25 b = 0.5;
26 c = 1;
27 d = 1;
28 psi = 0.5;
29
30 %X=[q,p]
31
32 g1 = @(X)X(2)-a+b*X(1);
33 g2 = @(X)X(2)-c-d*X(1)^psi;
34 dg1 = @(X)b;
35 dg2 = @(X)1;

```



```

36
37 g = {g1,g2};
38 dg = {dg1,dg2};
39
40 ini_val = [0.1;0.1];
41 eps = 0.00001;
42 del = 0.001;
43 max_it = 10000;
44 dampening = 1;
45 stop_crit = [eps,del,max_it];
46
47 X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);
48
49 %X=[p,q]
50 g1 = @(X)X(1)-a+b*X(2);
51 g2 = @(X)X(1)-c-d*X(2)^psi;
52 dg1 = @(X) 1;
53 dg2 = @(X)-d*psi*X(2)^(psi-1);
54
55 g = {g1,g2};
56 dg = {dg1,dg2};
57 X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);
58
59 dampening = 0.5;
60 X = gauss_seidel(g,dg,ini_val,stop_crit,dampening);

```

Question 3

```

1  clc;clear
2
3  %% 3.1
4  % load data
5  [data txt]= xlsread('MRW92QE-data.xls');
6  data(:,2) = [];
7
8  % delete countries with missing data
9  No_Countries = size(data,1);
10 index = [];
11 for iCountry = 1:No_Countries
12     if sum(isnan(data(iCountry,:)))>0
13         index=[index iCountry];

```

```

14     end
15 end
16 data(index,:) = [];
17
18 %% 3.2
19 index_nonoil = [];
20 index_inter = [];
21 index_oecd = [];
22
23 for iCountry = 1:length(data)
24     % non-oil countries
25     if data(iCountry,2)==1
26         index_nonoil = [index_nonoil iCountry];
27     end
28     % intermediate countries
29     if data(iCountry,3)==1
30         index_inter = [index_inter iCountry];
31     end
32     % oecd countries
33     if data(iCountry,4)==1
34         index_oecd = [index_oecd iCountry];
35     end
36
37 end
38
39 nonoil_country = data(index_nonoil,:);
40 inter_country = data(index_inter,:);
41 oecd_country = data(index_oecd,:);
42
43 %% 3.3
44 %nonoil
45 gdp60_nonoil = log(nonoil_country(:,5));
46 gdp85_nonoil = log(nonoil_country(:,6));
47 iy_nonoil = log(nonoil_country(:,9));
48 school_nonoil = log(nonoil_country(:,10));
49
50 y_nonoil = gdp85_nonoil-gdp60_nonoil;
51 X_nonoil = [ones(size(y_nonoil,1),1) gdp60_nonoil iy_nonoil ...
52     log(nonoil_country(:,8)+5) school_nonoil];
53 Beta_nonoil = (X_nonoil'*X_nonoil)\(X_nonoil'*y_nonoil);
54 sigma_nonoil = var(y_nonoil-X_nonoil*Beta_nonoil)*...

```

```

55      (X_nonoil'*X_nonoil/size(y_nonoil,1))^(−1)./size(y_nonoil,1);
56
57 %intermediate country
58 gdp60_inter = log(inter_country(:,5));
59 gdp85_inter = log(inter_country(:,6));
60 iy_inter = log(inter_country(:,9));
61 school_inter = log(inter_country(:,10));
62
63 y_inter = gdp85_inter−gdp60_inter;
64 X_inter = [ones(size(y_inter,1),1) gdp60_inter iy_inter ...
65      log(inter_country(:,8)+5) school_inter];
66 Beta_inter = (X_inter'*X_inter)\(X_inter'*y_inter);
67 sigma_inter = var(y_inter−X_inter*Beta_inter)*...
68      (X_inter'*X_inter/size(y_inter,1))^(−1)./size(y_inter,1);
69
70 %oecd country
71 gdp60_oecd = log(oecd_country(:,5));
72 gdp85_oecd = log(oecd_country(:,6));
73 iy_oecd = log(oecd_country(:,9));
74 school_oecd = log(oecd_country(:,10));
75
76 y_oecd = gdp85_oecd−gdp60_oecd;
77 X_oecd = [ones(size(y_oecd,1),1) gdp60_oecd iy_oecd ...
78      log(oecd_country(:,8)+5) school_oecd];
79 Beta_oecd = (X_oecd'*X_oecd)\(X_oecd'*y_oecd);
80 sigma_oecd = var(y_oecd−X_oecd*Beta_oecd)*...
81      (X_oecd'*X_oecd/size(y_oecd,1))^(−1)./size(y_oecd,1);
82
83 %% 3.4
84
85 R2_nonoil = 1 − var(y_nonoil−mean(y_nonoil))\var(y_nonoil−X_nonoil*
      Beta_nonoil)...
86      *((length(y_nonoil)−1)/(length(y_nonoil)−1−length(Beta_nonoil)));
87 R2_inter = 1 − var(y_inter−mean(y_inter))\var(y_inter−X_inter*Beta_inter)
      ...
88      *((length(y_inter)−1)/(length(y_inter)−1−length(Beta_inter)));
89 R2_oecd = 1 − var(y_oecd−mean(y_oecd))\var(y_oecd−X_oecd*Beta_oecd)...
90      *((length(y_oecd)−1)/(length(y_oecd)−1−length(Beta_oecd)));
91
92
93 clearvars −except Beta* R2* sigma*

```

```

94
95 Beta = [Beta_nonoil Beta_inter Beta_oecd];
96 Std_Beta = [(diag(sigma_nonoil)).^0.5 (diag(sigma_inter)).^0.5 (diag(
    sigma_oecd)).^0.5];
97 R2 = [R2_nonoil R2_inter R2_oecd];

```

Question 4

```

1  % Computational Economics PS2 Q4
2
3  clear , clc
4  close all
5
6  %% Parameter and function initialization
7  alpha_k = 0.33;
8  alpha_h = 0.33;
9  s_k = 0.2;
10 s_h = 0.2;
11 delta_k = 0.1;
12 delta_h = 0.06;
13 n = 0.01;
14 g = 0.015;
15 ini_val = [3,3];
16 ini_Jac = eye(2);
17
18 fun = @(val) steady_state(val, alpha_k, alpha_h, s_k, s_h, ...
19                          delta_k, delta_h, n, g);
20 Jac = @(val) Jacobian(val, alpha_k, alpha_h, s_k, s_h, ...
21                      delta_k, delta_h, n, g);
22 fun_fp = @(val) steady_state_fixed_pt(val, alpha_k, alpha_h, s_k, s_h, ...
23                                       delta_k, delta_h, n, g);
24
25 %% Q4.1
26 k_star = (s_k/(n+g+delta_k)) ^ ((1-alpha_h)/(1-alpha_k-alpha_h)) * ...
27          (s_h/(n+g+delta_h)) ^ ((alpha_h)/(1-alpha_k-alpha_h));
28 h_star = (s_h/(n+g+delta_h)) ^ ((1-alpha_k)/(1-alpha_k-alpha_h)) * ...
29          (s_k/(n+g+delta_k)) ^ ((alpha_k)/(1-alpha_k-alpha_h));
30 x_star = [k_star; h_star];
31
32 %% Q4.3
33 eps = 0.001;

```

```
34 del = 0.001;
35 max_it = 1000;
36 stop_crit = [eps, del, max_it];
37
38 x_hat = Newton_Method(fun, Jac, ini_val, stop_crit);
39 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
40
41 eps = 1.e-20;
42 del = 1.e-20;
43 max_it = 1000;
44 stop_crit = [eps, del, max_it];
45
46 x_hat = Newton_Method(fun, Jac, ini_val, stop_crit);
47 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
48
49 %% Q4.4
50 eps = 0.001;
51 del = 0.001;
52 max_it = 1000;
53 stop_crit = [eps, del, max_it];
54
55 x_hat = Broyden_Method(fun, ini_Jac, ini_val, stop_crit);
56 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
57
58 eps = 1.e-20;
59 del = 1.e-20;
60 max_it = 1000;
61 stop_crit = [eps, del, max_it];
62
63 x_hat = Broyden_Method(fun, ini_Jac, ini_val, stop_crit);
64 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
65
66 %% Q4.5
67 eps = 0.001;
68 del = 0.001;
69 max_it = 1000;
70 stop_crit = [eps, del, max_it];
71
72 x_hat = Inverse_Broyden_Method(fun, ini_Jac, ini_val, stop_crit);
73 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
74
```

```

75 eps = 1.e-20;
76 del = 1.e-20;
77 max_it = 1000;
78 stop_crit = [eps, del, max_it];
79
80 x_hat = Inverse_Broyden_Method(fun, ini_Jac, ini_val, stop_crit);
81 disp(['Inf norm = ', num2str(max(abs(x_hat-x_star)))])
82
83 %% Q4.6
84 eps = 1.e-10;
85 del = 0.001;
86 max_it = 10000;
87 stop_crit = [eps, del, max_it];
88
89 [x_hat, x_history] = Fixed_Point_Method(fun_fp, ini_val, stop_crit, true);
90
91 figure
92 subplot(2,1,1)
93 plot(1:size(x_history,2), x_history(1,:))
94 hold on
95 plot([1, size(x_history,2)], [k_star, k_star], 'k')
96 xlim([1, size(x_history,2)])
97 title('k')
98
99 subplot(2,1,2)
100 plot(1:size(x_history,2), x_history(2,:))
101 hold on
102 plot([1, size(x_history,2)], [h_star, h_star], 'k')
103 xlim([1, size(x_history,2)])
104 title('h')

```

Question 5

```

1 function Q5
2 clc; clear;
3
4 EquilibriumFunc = @(q) func(q, 1.6);
5
6 n=[2;5;10];
7
8 for i =1:length(n)

```

```

9
10 ini_Jac=eye(n(i));
11 ini_val = ones(n(i),1);
12
13 eps = 1.e-10;
14 del = 1.e-10;
15 max_it = 10000;
16 stop_crit = [eps, del, max_it];
17
18 q = Broyden_Method(EquilibriumFunc, ini_Jac, ini_val, stop_crit);
19 disp(['Equilibrium Output q = ', num2str(q')]);
20 end
21 end
22
23 function f = func( q, lambda )
24 % This function gives the equilibrium output levels.
25
26 n=length(q);
27 xi = 0.6;
28 f = zeros(n,1);
29
30 for i = 1:n
31     xi = xi+(i-1)*0.2/(n-1);
32     f(i) = sum(q)^(-1/lambda)-1/lambda*sum(q)^(-1/lambda-1)*q(i)-xi*q(i);
33 end
34
35 end

```

Functions

Question 2

```

1 function [x fx] = bisection(f,x,cc )
2 % This function calculate the root for f using bisection method.
3
4 x1 = x(1,1);
5 xh = x(2,1);
6 fl = f(x1);
7 fh = f(xh);
8

```

```

9  tole = cc(1,1);
10 told = cc(2,1);
11
12 if fl*fh>0
13     disp('initial [xl, xh] don''t bracket a root');
14     x = -Inf; fx = -Inf; ef = 0;
15     return
16 end
17
18 % bisection
19 while (xh-xl)>tole*(1+abs(xl)+abs(xh)) || abs(fm)>told
20     xm = (xl+xh)/2;
21     fm = f(xm);
22     if fl*fm < 0
23         xh = xm; fh = fm;
24     else
25         xl = xm; fl = fm;
26     end
27
28 end
29 x = xm; fx = fm;
30 end

```

Question 4

```

1  function roots = Broyden_Method(func, ini_Jac, ...
2                                ini_val, ...
3                                stop_crit)
4  %This function perform the Newton's method for root-finding problem.
5  %      func: a function handle for value of the root-finding problem.
6  %      Jac: a function handle of the Jacobian function of the problem
7  %      ini_val: initial value
8  %      stop_crit: stopping criteria = [eps, del, max_it]
9
10 eps = stop_crit(1);
11 del = stop_crit(2);
12 max_it = stop_crit(3);
13 it = 0;
14 cont = true;
15
16 if length(ini_val)~=size(ini_val,2)

```



```

17     xold = ini_val';
18 else
19     xold = ini_val;
20 end
21
22 J = ini_Jac;
23 fold = func(xold);
24
25 while cont
26     it = it+1;
27
28     xnew = xold-J\fold;
29     fnew = func(xnew);
30     dx = xnew-xold;
31     J = J+((fnew-fold-J*(dx))*dx')/(dx'*dx);
32     if (norm(xold-xnew)<=eps*(1+norm(xnew))) || (it==max_it)
33         cont = false;
34     end
35     xold = xnew;
36     fold = fnew;
37 end
38
39 if norm(func(xnew))<=del
40     disp(['Convergence after ',num2str(it),' iterations.'])
41     roots = xnew;
42 else
43     disp('Convergence failed.')
44     roots = [];
45 end
46
47 end

1 function roots = InverseBroydenMethod(func,ini_Jac,...
2                                     ini_val,...
3                                     stop_crit)
4 %This function perform the Newton's method for root-finding problem.
5 %     func: a function handle for value of the root-finding problem.
6 %     Jac: a function handle of the Jacobian function of the problem
7 %     ini_val: initial value
8 %     stop_crit: stopping criteria = [eps,del,max_it]
9
10 eps = stop_crit(1);

```

```

11 del = stop_crit(2);
12 max_it = stop_crit(3);
13 it = 0;
14 cont = true;
15
16 if length(ini_val)==size(ini_val,2)
17     xold = ini_val';
18 else
19     xold = ini_val;
20 end
21
22 B = inv(ini_Jac);
23 fold = func(xold);
24
25 while cont
26     it = it+1;
27
28     xnew = xold-B*fold;
29     fnew = func(xnew);
30     dx = xnew-xold;
31     df = fnew-fold;
32     B = B+((dx-B*df)*dx'*B)/(dx'*B*df);
33     if (norm(xold-xnew)<=eps*(1+norm(xnew))) || (it==max_it)
34         cont = false;
35     end
36     xold = xnew;
37     fold = fnew;
38 end
39
40 if norm(func(xnew))<=del
41     disp(['Convergence after ',num2str(it),' iterations.'])
42     roots = xnew;
43 else
44     disp('Convergence failed.')
45     roots = [];
46 end
47
48 end

1 function J = Jacobian(val,alpha_k,alpha_h,s_k,s_h,...
2                     delta_k,delta_h,n,g)
3 %This function calculate the Jacobian of the human capital augmented Solow

```

```

4  %growth model.
5
6  k = val(1);
7  h = val(2);
8
9  J = ...
10     [ alpha_k*s_k*k^(alpha_k-1)*h^alpha_h-(n+g+delta_k) , ...
11       alpha_h*s_k*k^alpha_h*h^(alpha_h-1); ...
12       alpha_k*s_h*k^(alpha_k-1)*h^alpha_h , ...
13       alpha_h*s_h*k^alpha_h*h^(alpha_h-1)-(n+g+delta_h) ];
14
15  end

1  function roots = NewtonMethod(func, Jac, ...
2                               ini_val, ...
3                               stop_crit)
4  %This function perform the Newton's method for root-finding problem.
5  %      func: a function handle for value of the root-finding problem.
6  %      Jac: a function handle of the Jacobian function of the problem
7  %      ini_val: initial value
8  %      stop_crit: stopping criteria = [eps, del, max_it]
9
10  eps = stop_crit(1);
11  del = stop_crit(2);
12  max_it = stop_crit(3);
13  it = 0;
14  cont = true;
15
16  if length(ini_val) == size(ini_val, 2)
17      xold = ini_val';
18  else
19      xold = ini_val;
20  end
21
22  while cont
23      it = it+1;
24      xnew = xold - Jac(xold)\func(xold);
25      if (norm(xold-xnew) <= eps*(1+norm(xnew))) || (it == max_it)
26          cont = false;
27      end
28      xold = xnew;
29  end

```

```

30
31 if norm(func(xnew))<=del
32     disp(['Convergence after ',num2str(it),' iterations.'])
33     roots = xnew;
34 else
35     disp('Convergence failed.')
36     roots = [];
37 end
38
39 end

1 function f = steady_state_fixed_pt(val,alpha_k,alpha_h,s_k,s_h,...
2     delta_k,delta_h,n,g)
3 %This function compute the value of the functions characterizing the steadt
4 %state of the human capital augmented Solow grotwh model.
5
6 k = val(1);
7 h = val(2);
8 y = k^alpha_k*h^alpha_h;
9
10 f = [...
11     s_k*y-(n+g+delta_k)*k;...
12     s_h*y-(n+g+delta_h)*h];
13
14 end

1 function f = steady_state(val,alpha_k,alpha_h,s_k,s_h,...
2     delta_k,delta_h,n,g)
3 %This function compute the value of the functions characterizing the steadt
4 %state of the human capital augmented Solow grotwh model.
5
6 k = val(1);
7 h = val(2);
8 y = k^alpha_k*h^alpha_h;
9
10 f = [...
11     s_k*y-(n+g+delta_k)*k;...
12     s_h*y-(n+g+delta_h)*h];
13
14 end

```