Computational Economics Problem Set 3

Names: Di Lu(5706079),), Zexi Sun(5917910), Penghui Yin(5702358), Shiwei Yu(5913037)

**Problem 1**

See code appendix PS3q1&Newton

**Problem 2**

2.1

Finite period and determined wage process, so we can use backward induction to solve this.

If t=T, Ct=at(1+r)+wt,, as aT+1=0,for t=T-1:-1:1, Ct=at(1+r)+wt+at+1.

Plug in to the Euler Equation uc,t=(1+r)βuc,t+1, we can get

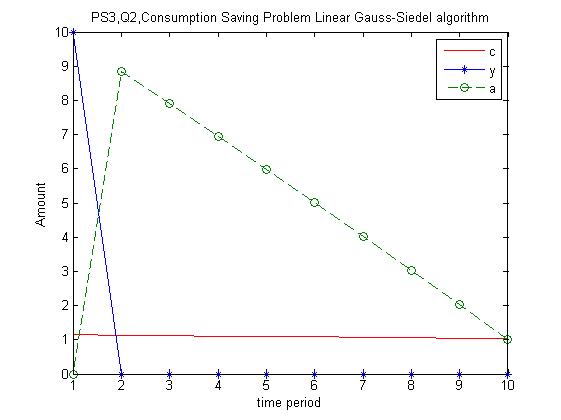
Then we get a series function, aT(aT-1,wT-1),aT-1(aT-2,wT-2),…,a2(a1, w2), accordingly, we get the consumption series via the budget constraint.

2.2

Parameters beta=0.95;r=0.02;theta=3;T=10;max=1000;tol=1e-30;

We take a in each period as a variable, and then they form a multi-nonlinear system equation.

We both apply the Gauss-Jacobi algorithm and Gauss-Siedel algorithm, and the get the same result, so we just show the picture once. Furthermore, the time for Gauss-Jacobi algorithm is 0.044528 second and the time for Gauss-Siedel algorithm is 0.010121 second, so the latter one is indeed quicker. Code see appendix PS3q2



2.3

In the case of our algorithm, we don’t need to worry about if theta=1, because we already use the first order condition. If we do it in another iteration, we just use the IF logic. That is, if theta=1, the utility function is log function, otherwise, it is a CRRA function.

2.4

One way is to use the punishment term. If c is negative, we set the utility to negative infinite manually. Or we can transfer the c in another form, say d to the power of c, which d is a positive large number. So if c is negative, d is small, though not changing as much as the first method.

Code appendix

PS3 q1

|  |
| --- |
| close all;  clear all  clc;  %% PS3 q1 Newton Method  cc=[eps,eps,100]';x0=2;    syms x y ;  y = 2\*x^3-x^2-3\*x+2;  dy=diff(y);  ddy=diff(dy);  f=matlabFunction(y,dy,ddy);  txt='y = 2\*x^3-x^2-3\*x+2';  fsol=Newton(f,x0,cc,txt);    disp(' ')    y=-x\*exp(-x);  dy=diff(y);  ddy=diff(dy);  g=matlabFunction(y,dy,ddy);  txt='y = -x\*exp(-x)';  gsol=Newton(g,x0,cc,txt);    % solution with Matlab solver:  y=2\*x^3-x^2-3\*x+2;  f1= matlabFunction(y);  xmnew\_f = fminunc(f1,x0);    y=-x\*exp(-x);  g1= matlabFunction(y);  xmnew\_g = fminunc(g1,x0); |

Newton

|  |
| --- |
| function [x,fx,ef] = Newton(f,x,cc,t)  tole = cc(1,1); told = cc(2,1); maxiter = cc(3,1);  ef = 0;x0=x;  for j = 1:maxiter  [fx,dfx,ddfx] = f(x);  xp = x - ddfx\dfx';  if norm(x-xp) <= tole\*(1+norm(xp))  ef = 2; break; % converged in first criterion  else  x = xp;  end  end  if norm(dfx) <= tole\*(1+norm(fx))  if ef == 2; ef=1; % converged in both criteria  else ef=3; % spurious convergence  end  end  sol=[ef,x,fx];  fprintf('for starting value %4.2f of this funtion %s',x0,t)  disp('')  fprintf(' the solution indicator is %1.0f,solution x is %8.3f, function value fx is %8.3f', sol)  disp('') |

PS3 q2

|  |
| --- |
| clc; close all; clear all;  %%Multi equation  %set parameters and variables  beta=0.95;r=0.02;theta=3;T=10;max=1000;tol=1e-30;  c=ones(1,T);a0=c;w=zeros(1,T);w(1)=10;t=(1:1:T);a0(1)=0;  %a(1)=0,a(t+1)=0,a can't be set as all zeros in the first time    %(a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta)-beta\*(1+r)\*(a(T)\*(1+r)+w(T))^(-theta)=0;  %FOC for T-1  %further with respect to a(T)  %theta\*(a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(T)\*(1+r)+w(T))^(-theta-1)=0  %for i=T-2:-1:1 (a(i)\*(1+r)+w(i)-a(i+1))^(-theta)-beta\*(1+r)\*(a(i+1)\*(1+r)+w(i+1)-a(i+2))^(-theta)=0  %rest of the period  %further with respect to a(i+1)  %theta\*(a(i)\*(1+r)+w(i)-a(i+1))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(i+1)\*(1+r)+w(i+1)-a(i+2))^(-theta-1)=0    %Linear Gauss-Jacobi algorithm:  %xi\_k+1=xi\_k-gi(xi\_k)/gi'(xi\_k)£¬no updating further  a=a0;a1=a0;  tic;  for j=1:max  for i=T-1:-1:2  a1(T)=a(T)-((a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta)-beta\*(1+r)\*(a(T)\*(1+r)+w(T))^(-theta))/(theta\*(a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(T)\*(1+r)+w(T))^(-theta-1));  a1(i)=a(i)-((a(i-1)\*(1+r)+w(i-1)-a(i))^(-theta)-beta\*(1+r)\*(a(i)\*(1+r)+w(i)-a(i+1))^(-theta))/(theta\*(a(i-1)\*(1+r)+w(i-1)-a(i))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(i)\*(1+r)+w(i)-a(i+1))^(-theta-1));  %a1(1)=a(1)-((-a(1))^(-theta)-beta\*(1+r)\*(w(1)-a(1))^(-theta))/(theta\*(-a(1))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(1)\*(1+r)+w(1)-a(2))^(-theta-1));  end  dist=abs(a1-a);  if (dist<tol),  break  else  a=a1;  end  end  t1=toc;a1  fprintf('the time for Gauss-Jacobi algorithm is %8.6f second',t1)  disp(' ')  %consumption setting;  c1=c;  for i=1:1:T-1;  c1(i)=a1(i)\*(1+r)+w(i)-a1(i+1);  end  c1(T)=a1(T)\*(1+r)+w(T);  c1  plot(t,c1,'r',t,w,'-\*',t,a1,'--o')  title('PS3,Q2,Consumption Saving Problem Linear Gauss-Jacobi algorithm')  xlabel('time period')  xlim([1 T])  ylabel('Amount')  legend('c','y','a','Location','NorthEast')    %Linear Gauss-Siedel algorithm:  %xi\_k+1=xi\_k-gi(xi\_k)/gi'(xi\_k)£¬no updating further  a=a0;a2=a0;  tic  for j=1:max  for i=T-1:-1:2  a(T)=a(T)-((a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta)-beta\*(1+r)\*(a(T)\*(1+r)+w(T))^(-theta))/(theta\*(a(T-1)\*(1+r)+w(T-1)-a(T))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(T)\*(1+r)+w(T))^(-theta-1));  a(i)=a(i)-((a(i-1)\*(1+r)+w(i-1)-a(i))^(-theta)-beta\*(1+r)\*(a(i)\*(1+r)+w(i)-a(i+1))^(-theta))/(theta\*(a(i-1)\*(1+r)+w(i-1)-a(i))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(i)\*(1+r)+w(i)-a(i+1))^(-theta-1));  %a1(1)=a(1)-((-a(1))^(-theta)-beta\*(1+r)\*(w(1)-a(1))^(-theta))/(theta\*(-a(1))^(-theta-1)+theta\*beta\*(1+r)^2\*(a(1)\*(1+r)+w(1)-a(2))^(-theta-1));  end  dist=abs(a2-a);  if (dist<tol),  break  else  a2=a;  end  end  t2=toc;a2  fprintf('the time for Gauss-Siedel algorithm is %8.6f second',t2)  %consumption setting;  disp(' ')  c2=c;  for i=1:1:T-1;  c2(i)=a2(i)\*(1+r)+w(i)-a2(i+1);  end  c2(T)=a2(T)\*(1+r)+w(T);  c2  figure  plot(t,c2,'r',t,w,'-\*',t,a2,'--o')  title('PS3,Q2,Consumption Saving Problem Linear Gauss-Siedel algorithm')  xlabel('time period')  xlim([1 T])  ylabel('Amount')  legend('c','y','a','Location','NorthEast') |

PS3 q3.3 q3.4 q3.5

global gam bet... % preference parameters

rk rh eta... % production parameters

dk dh... % depreciation rates

k % curvature of alpha functions

function [kt\_quation] = KTeqs(x,y)

globalvarKT

% control variables

c1 = x(1,1); c2 = x(2,1);

ik = x(3,1); ih = x(4,1);

alphak = x(5,1); alphah = x(6,1);

mu1 = -x(7,1); mu2 = -x(8,1);

% state variables

k1 = y(1,1); h1 = y(2,1);

% Kuhn-Tucker equations

kt\_quation(1,1) = c1^(-gam) + mu1;

kt\_quation(2,1) = bet \* c2^(-gam) + mu2;

kt\_quation(3,1) = max(alphak,0)^k + mu1 -mu2\*(1+rk);

kt\_quation(4,1) = max(alphah,0)^k + mu1 -mu2\*rh\*eta\*((1-dh)\*h1+ih)^(eta-1);

kt\_quation(5,1) = max(-alphak,0)^k -((1-dk)\*k1 + ik);

kt\_quation(6,1) = max(-alphah,0)^k -ih;

kt\_quation(7,1) = c1 - (rk\*k1 + rh\*h1 - ik - ih);

kt\_quation(8,1) = c2 - ((1+rk)\*((1-dk)\*k1 + ik) + rh\*((1-dh)\*h1+ih)^(eta));

clear all

close all

clc

%% Parameterization

globalvarKT

gam = 2; % coefficient of relative risk aversion

bet = 0.96; % time preference

rk = 0.1; % aggregate return to physical capital

rh = 1.4; % aggregate return to human capital

eta = 0.8; % curvature of individual return to human capital

dk = 0.05; % depreciation rate on physical capital

dh = 0.05; % depreciation rate on human capital

k = 2; % curvature of alpha functions

options = optimset('Display','none', 'MaxIter', 2000, 'MaxFunEvals', 2000);

%% Solve Model

xini = [1;1;1;1;1;1;1;1];

y1 = [1; 5];

f1 = @(x) KTeqs(x,y1);

[x1, fv1, ef1] = fsolve(f1, xini, options);

if (ef1 == 1 && isreal(x1))

converged='fsolve converged';

elseif (ef1 ~= 1)

converged = ['ERROR: fsolve didn"t converge, exitflag = ',num2str(ef1)];

else

converged = ['ERROR: solution contains imaginary part, exitflag = ',num2str(ef1)];

end

k2= (1-dk) \*y1(1) + x1(3);

fprintf('%s k1 = %7.4f; h1 = %7.4f; %s \n', 'Initial values:', y1', converged)

fprintf('Optimal choices: c1 = %7.4f; c2 = %7.4f; ik = %7.4f; ih = %7.4f; k2 = %7.4f \n', x1(1:4,1)', k2)

if (x1(4) == 0 && k2 == 0)

disp('Irreversibility and borrowing constraints bind')

elseif (x1(4) == 0 )

disp('Only irreversibility constraint binds')

elseif (k2 == 0)

disp('Only borrowing constraint binds')

end

disp(' ');

y2 = [1; 1];

f2 = @(x) KTeqs(x,y2);

[x2, fv2, ef2] = fsolve(f2, xini, options);

if (ef2 == 1 && isreal(x2))

converged='fsolve converged';

elseif (ef2 ~= 1)

converged = ['ERROR: fsolve didn"t converge, exitflag = ',num2str(ef2)];

else

converged = ['ERROR: solution contains imaginary part, exitflag = ',num2str(ef2)];

end

k2= (1-dk) \*y2(1) + x2(3);

fprintf('%s k1 = %7.4f; h1 = %7.4f; %s \n', 'Initial values:', y2', converged)

fprintf('Optimal choices: c1 = %7.4f; c2 = %7.4f; ik = %7.4f; ih = %7.4f; k2 = %7.4f \n', x2(1:4,1)', k2)

if (x2(4) == 0 && k2 == 0)

disp('Irreversibility and borrowing constraints bind')

elseif (x2(4) == 0 )

disp('Only irreversibility constraint binds')

elseif (k2 == 0)

disp('Only borrowing constraint binds')

else

disp('No constraint binds')

end

disp(' ');

y3 = [1; 0.2];

f3 = @(x) KTeqs(x,y3);

[x3, fv3, ef3] = fsolve(f3, xini, options);

if (ef3 == 1 && isreal(x3))

converged='fsolve converged';

elseif(ef3 ~= 1)

converged = ['ERROR: fsolve didn"t converge, exitflag = ',num2str(ef3)];

else

converged = ['ERROR: solution contains imaginary part, exitflag = ',num2str(ef3)];

end

k2= (1-dk) \*y3(1) + x3(3);

fprintf('%s k1 = %7.4f; h1 = %7.4f; %s \n', 'Initial values:', y3', converged)

fprintf('Optimal choices: c1 = %7.4f; c2 = %7.4f; ik = %7.4f; ih = %7.4f; k2 = %7.4f \n', x3(1:4,1)', k2)

if (x3(4) == 0 && k2 == 0)

disp('Irreversibility and borrowing constraints bind')

elseif (x3(4) == 0 )

disp('Only irreversibility constraint binds')

elseif (k2 == 0)

disp('Only borrowing constraint binds')

end

disp(' ');

%% for the optimal human/physical capital ratio

for i=0:0.01:5;

y4 = [1; i];

f4 = @(x) KTeqs(x,y4);

[x4, fv4, ef4] = fsolve(f4, xini, options);

if x4(4,1)==0

fprintf('Optimal physical to human captial ratio = 1 : %7.4f \n', i)

break

end

end

PS3 q4.2 q4.3

syms a;

rf = 0.02; rlr = -0.08-rf; rhr = 0.12-rf;

p = 0.1; phi = -3;

f = -(p\*(1 + rf + a\*rlr)^phi + (1 - p)\*(1 + rf + a\*rhr)^phi)/phi;

df = diff(f,a);

mf = matlabFunction(f);

mdf = matlabFunction(df);

astar1 = fzero(mdf,1);

display(['The unconstrained solution is ', num2str(astar1)]);

figure;

set(gcf,'color','w');

ezplot(-f,[astar1-1,astar1+1]);

xlabel('value of alpha');

ylabel('value of the objective (w0=1)');

title('');

print('plot42','-dpng')

astar2 = fminbnd(mf,0,1);

astar3 = fmincon(mf,0.5,[],[],[],[],0,1);

display(['The function fminbnd and fmincon give the constrained solution ', num2str(astar2), ' and ', num2str(astar3)]);