# ECON 6140 - Problem Set # 3

Julien Manuel Neves

April 20, 2018

Buckle up! This problem set as a lot of code and very few comments in it. I would have loved to make it neater, but I ran out of time and .

## Problem #1

(a) See `main.m`, `tauchen.m`, and `transition.m`. Moreover, see Tab b. Note that $\mathsf{E}\{y_t\} = e^{\mu + \frac{\sigma^2}{2}}$ where $\mu$ and $\sigma^2$ are the expected moments of $w_t$. Hence, for $\mathsf{E}\{y_t\} = 1$ ,we need

$$\bar{w} = -(1-\rho)\frac{\sigma_\epsilon^2}{2(1-\rho^2)}$$

(b) See `main.m`, `tauchen.m`, `rouwenhorst.m`, and `transition.m`. The results are reported in Tab b. It seems that the Tauchen method does fairly well to approximate $\rho$ and $\sigma_\epsilon$. Note that increasing the number of states seems to have a ambiguous effectiveness. My guess is that with an even number of states, we are indirectly missing the point with the highest probability mass, i.e. $\mu$. Hence, 9 states could potential do better than only 5 states or even 10 states.

| | $\rho$ | $\sigma_\epsilon$ |
|---|---|---|
| Model | .90 | .24495 |
| Tauchen (5 points) | .92665 | .26813 |
| Tauchen (10 points) | .89168 | .27535 |

Table 1: Tauchen with 5 points and 10 points

(c) See `main.m`, `tauchen.m`, `rouwenhorst.m`, and `transition.m`. The results are reported in Tab c. Clearly, the Rouwenhorst is yields the best approximation of $\rho$ and $\sigma_\epsilon$ since it perfectly matches the parameters.

| | $\rho$ | $\sigma_\epsilon$ |
|---|---|---|
| Model | .90 | .24495 |
| Tauchen (5 points) | .92665 | .26813 |
| Rouwenhorst (5 points) | .90 | .24495 |

Table 2: Tauchen with 5 points and Rouwenhorst 5 points

(d) See `main.m`, `tauchen.m`, `rouwenhorst.m`, and `transition.m`. The results are reported in Tab d. Lo, and behold! Even with $\rho$ close to 0, the Rouwenhorst perfectly matches $\rho$ and $\sigma_\epsilon$.

| | $\rho$ | $\sigma_\epsilon$ |
|---|---|---|
| Model | .98 | .012505 |
| Rouwenhorst (5 points) | .98 | .012505 |

Table 3: Rouwenhorst 5 points and high $\rho$

Note that to match $var(w_t)$, we need to decrease $\sigma_\epsilon$. In fact, we need

$$\sigma_\epsilon^2 = \frac{\sigma_{\epsilon,old}^2(1-\rho^2)}{(1-\rho_{old}^2)}$$

## Problem #2

(a) I modified Prof. Huckfedlt's code to get rid of the pesky globals and try to reduce everything to functions. See `main.m`, `policy_ip.m`, `euler_ip.m`, and `solve_ip.m`. Fig a and Fig a chose the decision rules for consumption and $a'(a, y)$ respectively.

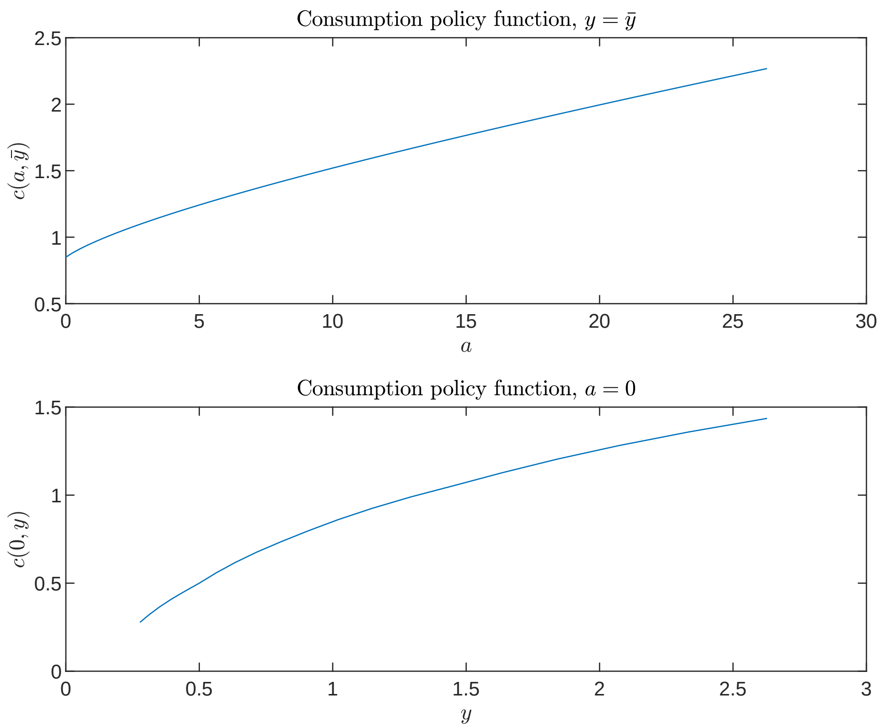



Figure 1: Consumption policy function

Note that for low $y$ and $a = 0$, the borrowing constraints binds and we get the bent shape in our savings policy function.

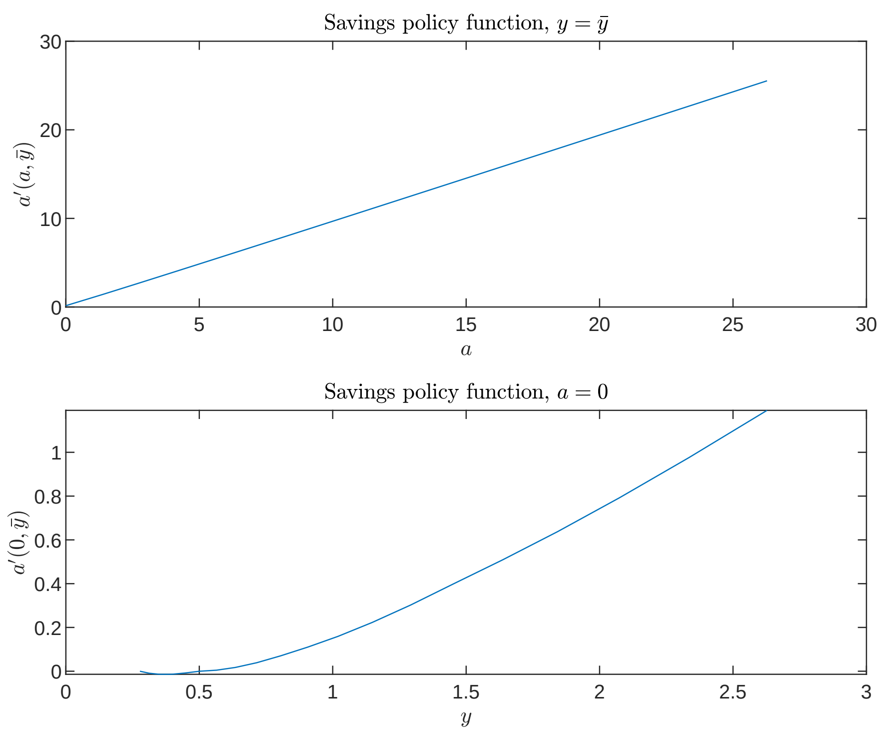



Figure 2: Savings policy function

(b) See `markovchain.m`, and `markovprob.m` for the Markov chain simulation. The results for the unconditional standard deviation of $c$ are reported in Tab b.

| | $\sigma_c$ |
|---|---|
| $\gamma = 1$ | .49118 |
| $\gamma = 2$ | .44302 |
| $\gamma = 5$ | .39500 |

Table 4: $\sigma_c$ for different values of $\gamma$

(c) Fig c plots the saving rates given $a = 0$ for different $y$.

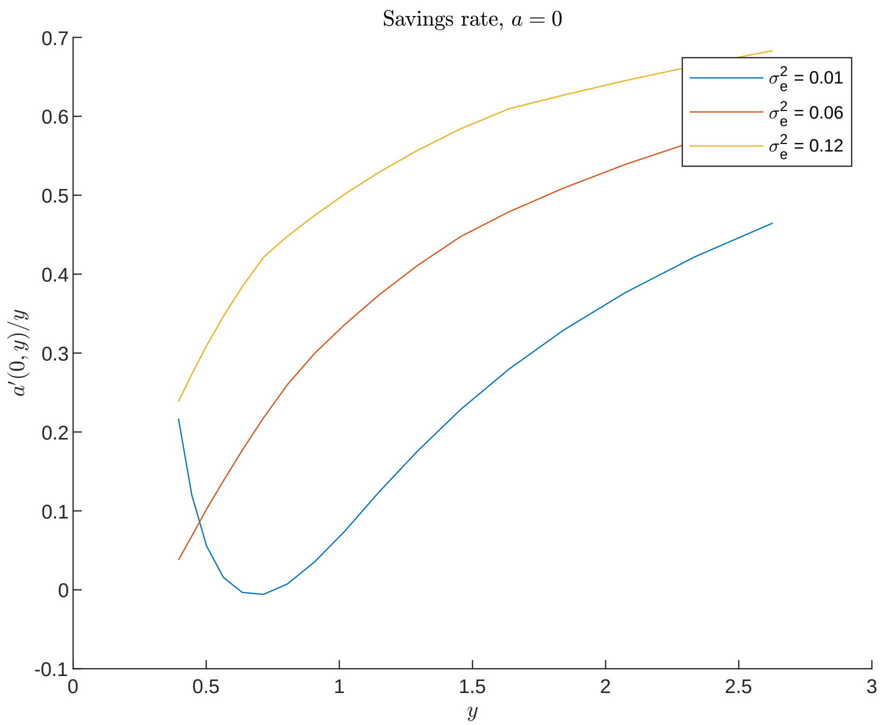



Figure 3: Savings rate

Note that as $\sigma_\epsilon$ increases, the savings rate also increases for all possible $y$. Hence, $\sigma_\epsilon$ increasing, implies higher volatility in income and therefore higher motive for precautionary savings. Please disregard the left part of the curve for $\sigma_\epsilon^2 = 0.01$. This is result is simply a consequence of how our grid was set up.

(d) The results for the mean $c$ with different borrowing constraints are reported in Tab d.

| | $\bar{c}$ |
|---|---|
| $a \geq 0$ | 1.0571 |
| $a \geq -\frac{y_{min}}{r}$ | .92897 |

Table 5: $\bar{c}$ for different borrowing constraints

It seems that having a looser borrowing constraint, implies lower average consumption. This is the results of having less motive for precautionary savings. Hence, by saving less, people get less consumption later on. While the value of utility might be higher due to the timing of consumption with the natural borrowing constraint, we get still get that on average consumption is lower than with no-borrowing.

(e) The results for the insurance coefficient with different borrowing constraints are reported in Tab 6.

| | $\psi$ |
|---|---|
| $a \geq 0$ | .56445 |
| $a \geq -\frac{y_{min}}{r}$ | .57244 |

Table 6: $\psi$ for different borrowing constraints

Note that having a looser borrowing constraint, implies higher insurance coefficient. In fact, if $\epsilon_t$ and $c_t$ are more correlated, then shocks to income will directly transfer to shocks to consumption. This in turns will decrease $\psi$, since the $c_t$ are not protected against income volatility. Hence, natural borrowing constraint with its higher $\psi$ implies that people are not a scared of shocks that could reduce their income since they are basically never restricted in their borrowing unlike $a \geq 0$.

# Problem #3

(a) Let $x_t = (1 + r_t)a_t + y_t$ or $x = Ra + y$. Then,

$$\begin{aligned} c + a' &\leq Ra + y \\ c + \frac{x' - y'}{R} &\leq x \\ Rc + x' - y' &\leq Rx \\ x' &\leq R(x - c) + y' \end{aligned}$$

and

$$
\begin{aligned}
a' &\geq 0 \\
\frac{x' - y'}{R} &\geq 0 \\
x' &\geq y' \\
x &\geq c
\end{aligned}
$$

Now, since our process is i.i.d., we have $\pi(y', y) = \pi(y')$. Hence, in this problem, $y$ is not a state variable, but it does influence the expected value of $x'$ since it is a function $y'$. Thus, our cash-on-hand problem can be summarized by

$$
\begin{aligned}
V(x) = \max_{c,x'} &\left\{ u(c) + \beta \sum_{y' \in Y} \pi(y') V(x') \right\} \\
\text{s. t. } \; x' &= R(x - c) + y' \\
x &\geq c
\end{aligned}
$$

Note that the Euler equation becomes

$$
u'(c) \geq \beta R \, \mathsf{E}_t \left\{ u'(c') \right\}
$$

where $x' \leq R(x - c) + y'$ and it holds with equality when $x = c$.

(b) Note that $\mathsf{E}\left\{y_t\right\} = e^{\mu + \frac{\sigma^2}{2}}$ where $\mu$ and $\sigma^2$ are the expected moments of $w_t$. Hence, for $\mathsf{E}\left\{y_t\right\} = 1$ ,we need

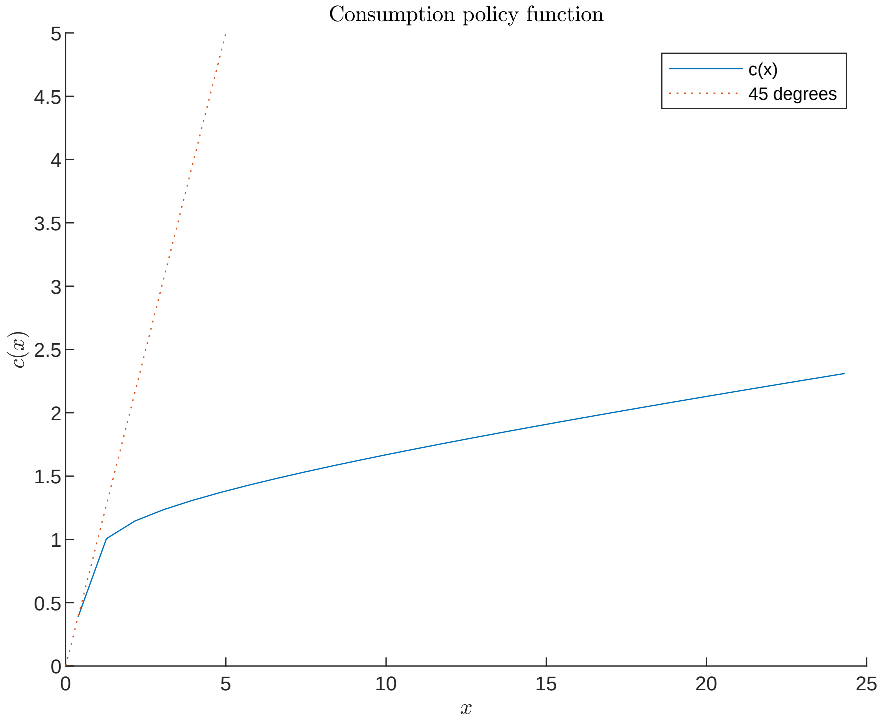
$$
\bar{w} = -\frac{\sigma_\epsilon^2}{2}
$$

Using `qnwnorm`, we get that this $\bar{w}$ does imply $\mathsf{E}\left\{y_t\right\} = 1$. See `main.m` for the code.

(c) See `main.m`, `policy_ca.m`, `euler_ca.m`, and `solve_ca.m` for the code. Note that I realized that there was a mistake in my code a bit too late to let the others know. I am truly sorry about that.

In this problem, we have only one state and that instead of using a transition matrix to compute the expected value, I decided to use the results from `qnwnorm` with 7 states from part (b) to compute the expectation in our Euler equation.

Fig c shows the policy function for $c(x)$.




Note that for low $x$, the borrowing constraint $x' \geq y$ is binding.

# Code

## main.m

```
clear; clc;
%% Problem 1
% declare parameters
param.ro=0.90;
param.se=sqrt(0.06);

% part a) - b)
param.k = 5;
[param, ro_ta5, se_ta5] = transition(param, "tauchen");
param.k = 10;
[param, ro_ta10, se_ta10] = transition(param,  "tauchen");

tab_1b = table([param.ro; ro_ta5; ro_ta10],[param.se; se_ta5;
    se_ta10],'VariableNames',{'rho','se'},'RowNames',{'Model';'Five
    ';'Ten'})

% part c)
param.k = 5;
```

```
[param, ro_rw, se_rw] = transition(param, "rouwenhorst");

tab_1c = table([param.ro; ro_ta5; ro_rw],[param.se; se_ta5; se_rw],'VariableNames',{'rho','se'},'RowNames',{'Model'; 'tauchen'; 'rouwenhorst'})

% part d)
old_var = param.se^2/(1-param.ro^2);
param.ro = 0.98;
param.se = old_var*(1-param.ro^2);
[param, ro_rw, se_rw] = transition(param, "rouwenhorst");

tab_1d = table([param.ro;ro_rw],[param.se;se_rw],'VariableNames',{'rho','se'},'RowNames',{'Model'; 'rouwenhorst'})

%% Problem 2
% declare parameters
param.beta=0.95;
param.r=0.02;
param.ro=0.9;
param.se=sqrt(0.06);
param.gamma=2;
param.amin = 0;
param.n = 25;
param.k = 5;

% part a)
[param] = transition(param, "rouwenhorst");
[param,c,fspace,s,smin,smax] = policy_ip(param);

close all
sfine=gridmake(nodeunif(param.n*2,smin(1),smax(1)),param.ygrid);
xfine=funeval(c,fspace,sfine);

figure(1)
subplot(2,1,1)
sfine=gridmake(nodeunif(param.n*4,smin(1),smax(1)),0); %ygrid(floor(k/2)+2));
xfine=funeval(c,fspace,sfine);
plot(sfine(:,1),xfine)
xlabel({'$a$'},'Interpreter','latex')
ylabel({'$c(a,\bar{y})$'},'Interpreter','latex')
title({'Consumption policy function, $y=\bar{y}$'},'Interpreter','latex')
set(gca,'FontSize',8);
```

```matlab

subplot(2,1,2)
sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));
xfine=funeval(c,fspace,sfine);
plot(exp(sfine(:,2)),xfine)
xlabel('$y$','Interpreter','latex')
ylabel('$c(0,y)$','Interpreter','latex')
title({'Consumption policy function, $a=0$'},'Interpreter','latex')
set(gca,'FontSize',8);
print -depsc fig1.eps

figure(2)
subplot(2,1,1)
sfine=gridmake(nodeunif(param.k*4,smin(1),smax(1)),0);
xfine=funeval(c,fspace,sfine);
plot(sfine(:,1),(1+param.r)*sfine(:,1)+exp(sfine(:,2))-xfine)
xlabel('$a$','Interpreter','latex')
ylabel('$a^{\prime}(a,\bar{y})$','Interpreter','latex')
title({'Savings policy function, $y=\bar{y}$'},'Interpreter','latex')
set(gca,'FontSize',8);

subplot(2,1,2)
sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));
xfine=funeval(c,fspace,sfine);
plot(exp(sfine(:,2)),(1+param.r)*sfine(:,1)+exp(sfine(:,2))-xfine)
xlabel('$y$','Interpreter','latex')
ylabel('$a^{\prime}(0,\bar{y})$','Interpreter','latex')
title({'Savings policy function, $a=0$'},'Interpreter','latex')
set(gca,'FontSize',8);
print -depsc fig2.eps

% part b)
gamma = [1,2,5];
for i = 1:3
    param.gamma = gamma(i);

    [param] = transition(param, "rouwenhorst");
    [param,c,fspace] = policy_ip(param);

    con = markovchain(param,c,fspace, 10000); % Generate Markov chain
    se_c(i) = std(con);
end
```

```
tab_2b = table(se_c','VariableNames',{'std_c'},'RowNames',{'Gamma
    = 1';'Gamma = 2';'Gamma = 5'})


% part c)
param.gamma = 2;
se = [0.01,0.06,0.12];

figure(3)
hold on
for i = 1:3
    param.se = sqrt(se(i));

    [param] = transition(param, "rouwenhorst");
    [param,c,fspace] = policy_ip(param);

    sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));
    xfine=funeval(c,fspace,sfine);
    plot(exp(sfine(4:end,2)),1-xfine(4:end)./exp(sfine(4:end,2)))
end
xlabel('$y$','Interpreter','latex')
ylabel('$a^{\prime}(0,y)/y$','Interpreter','latex')
title({'Savings rate, $a=0$'},'Interpreter','latex')
legend('\sigma_e^2 = 0.01','\sigma_e^2 = 0.06','\sigma_e^2 = 0.12'
    )
set(gca,'FontSize',8);
hold off
print -depsc fig3.eps


% part d) - e)
% no-borrowing
param.gamma = 2;
param.se = sqrt(0.06);
param.amin = 0;

[param] = transition(param, "rouwenhorst");
[param,c,fspace] = policy_ip(param);

[con, s] = markovchain(param,c,fspace, 10000);
avg_c(1) = mean(con);

c_t = log(con(2:end))-log(con(1:end-1));
e_t = s(2:end,2)-param.ro*s(1:end-1,2);
```

```
sig = cov(c_t,e_t);
phi(1) = 1-sig(1,2)/sig(2,2);

% natural debt limit
param.amin = -min(exp(param.ygrid)+.01)/param.r;

[param] = transition(param, "rouwenhorst");
[param,c,fspace] = policy_ip(param);

[con, s] = markovchain(param,c,fspace, 10000);
avg_c(2) = mean(con);

c_t = log(con(2:end))-log(con(1:end-1));
e_t = s(2:end,2)-param.ro*s(1:end-1,2);
sig = cov(c_t,e_t);
phi(2) = 1-sig(1,2)/sig(2,2);

tab_2d = table(avg_c',phi','VariableNames',{'mean_c','phi'},'
    RowNames',{'No borrowing';'Natural'})


%% Problem 3
% declare parameters
param.beta=0.95;
param.r=0.02;
param.ro=0;
param.se=sqrt(0.06);
param.gamma=2;
param.amin = 0;
param.n = 25;
param.k = 7;

% part b)
wbar = -param.se^2/2;
[x,w] = qnwnorm(7,wbar,param.se^2);
fprintf('E(y)=%f \n',exp(x)'*w)

% part c)
% [param] = transition(param, "rouwenhorst");
param.ws = w';
param.ygrid = x;
[param,c,fspace,s,smin,smax] = policy_ca(param);

figure(5)
hold on
```

```
sfine=gridmake(nodeunif(param.k*4,smin(1),smax(1)));
xfine=funeval(c,fspace,sfine);
plot(sfine,xfine)
xlabel('$x$','Interpreter','latex')
ylabel('$c(x)$','Interpreter','latex')
title({'Consumption policy function'},'Interpreter','latex')
set(gca,'FontSize',8);
plot([0:5],[0:5],':')
legend('c(x)','45 degrees')
hold off
print -depsc fig4.eps
```

**tauchen.m**

```
function [state_grid, prob] = tauchen(rho,sig_eps, ns, dev)
%markovchain Discretize AR(1) process
%   [prob, cum_prob, state_grid] = tauchen(ns, rho, sig_y ,dev)
    returns
%   prob, transition matrix,
%   cum_prob, cummulative distribution
%   state_grid, value of states

sig_y = sig_eps/sqrt(1-rho^2);  % Compute std of epsilon


state_grid = linspace(-dev*sig_y,dev*sig_y,ns)';    % Set state
    grid
d = mean(diff(state_grid));
% Compute y^j_t+1 - rho y^i_t
state_change = repmat(state_grid,1,ns)' - rho*repmat(state_grid
    ,1,ns);


prob = zeros(ns,ns);    % Initialize transition matrix
% Compute transition matrix according to Tauchen (1986)
prob(:,1) = cdf('norm',state_change(:,1)+d/2,0,sig_eps);
prob(:,end) = 1- cdf('norm',state_change(:,end) - d/2,0,sig_eps);
prob(:,2:end-1) = cdf('norm',state_change(:,2:end-1)+d/2,0,sig_eps
    )-cdf('norm',state_change(:,2:end-1)-d/2,0,sig_eps);

state_grid = state_grid';
end
```

**rouwenhorst.m**

```
%rouwenhorst.m
```

```
%
%[zgrid, P] = rouwenhorst(rho, sigma_eps, n)
%
% rho is the 1st order autocorrelation
% sigma_eps is the standard deviation of the error term
% n is the number of points in the discrete approximation
%
function [zgrid, P] = rouwenhorst(rho,sigma_eps,n)

mu_eps = 0;

q = (rho+1)/2;
nu = ((n-1)/(1-rho^2))^(1/2) * sigma_eps;

P = [q 1-q;1-q q];


for i=2:n-1
    P = q*[P zeros(i,1);zeros(1,i+1)] + (1-q)*[zeros(i,1) P;zeros(1,i+1)] + ...
        (1-q)*[zeros(1,i+1); P zeros(i,1)] + q*[zeros(1,i+1); zeros(i,1) P];
    P(2:i,:) = P(2:i,:)/2;
end

zgrid = linspace(mu_eps/(1-rho)-nu,mu_eps/(1-rho)+nu,n);
```

**transition.m**

```
function [param, ro_tilde, se_tilde] = transition(param, method)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

var_w= param.se^2/(1-param.ro^2);
wbar = -var_w*(1-param.ro)/2;
% nodes for the distribution of income shocks

if method=="tauchen"
    [e,w] = tauchen(param.ro,param.se, param.k, 3);
elseif method == "rouwenhorst"
    [e,w] = rouwenhorst(param.ro,param.se,param.k);        % Rouwenhorst method
end

param.yPP=w;
```

```
param.ygrid=e';

[~,D,V]=eig(w);
ws = V(:,1);
ws = ws/sum(ws);

param.ws = ws;
param.ygrid = param.ygrid + wbar/(1-param.ro);

ro_tilde = e*(w*e')/(e*e');
se_tilde = sqrt(e.^2*ws*(1-ro_tilde^2));
end
```

**policy_ip.m**

```
function [param,c,fspace,s,smin,smax] = policy_ip(param)
% Bounds for state space
ymin=min(param.ygrid);
ymax=max(param.ygrid);

amin = param.amin;                          % no borrowing
amax = 10*exp(ymax);                 % guess an upper bound on a, check later that do not exceed it

% Declare function space to approximate a'(a,y)
n=[param.n,param.k];

% Lower and higher bound for the state space (a,y)
smin=[amin,ymin];
smax=[amax,ymax];

scale=1/2;
fspace=fundef({'spli', nodeunif(n(1),(smin(1)-amin+.01).^scale,(smax(1)-amin+.01).^scale).^(1/scale)+amin-.01,0,3},...
    {'spli', param.ygrid,0,1});

grid=funnode(fspace);
s=gridmake(grid); %collection of  states (all a with y1... all a with y2... and so on)

c=funfitxy(fspace,s,param.r/(1+param.r)*s(:,1)+exp(s(:,2)));                %guess that keep constant assets

tic
for it=1:101
```

```
    cnew=c;
    x = solve_ip(param,c,fspace,s, amin);
    c=funfitxy(fspace,s,x);

    fprintf('%4i %6.2e\n',[it,norm(c-cnew)]);
    if norm(c-cnew)<1e-7, break, end
end
toc
end
```

**solve_ip.m**

```
function [x] = solve_ip(param,c, fspace, s, amin)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

ns=length(s);

a=.01*ones(ns,1);
b=(1+param.r)*s(:,1)+exp(s(:,2))-amin;
tol=1e-8; %tolerance level

fa=euler_ip(a,c,fspace,s,param);
fb=euler_ip(b,c,fspace,s,param);

x=zeros(ns,1);

% Start bisection
dx = 0.5*(b - a);

x = a + dx;                        %  start at midpoint
sb=sign(fb);
dx = sb.*dx;                       %  we increase or decrease x
    depending if f(b) is positive or negative

i=0;
  while any(abs(dx)>tol)
   i=i+1;
    dx = 0.5*dx;
    x = x - sign(euler_ip(x,c,fspace,s,param)).*dx;
  end


x(fb>=0)=b(fb>=0);
end
```

## euler_ip.m

```
function fval = euler_ip(x,c,fspace,s,param)

ns=size(s,1);
a=s(:,1);
y=exp(s(:,2));
aprime=(1+param.r)*a+y-x;
fval=x.^(-param.gamma);
as=ns/length(param.ygrid);

for i=1:length(param.ygrid)
  cprime=funeval(c,fspace,[aprime,param.ygrid(i)*ones(ns,1)]);
  for j=1:length(param.ygrid)
    ypp(1+(j-1)*as:j*as,1)=param.yPP(j,i);
  end
  fval=fval-param.beta*(1+param.r)*ypp.*cprime.^(-param.gamma);
end
```

## policy_ca.m

```
function [param,c,fspace,s,smin,smax] = policy_ca(param)
% Bounds for state space
ymin=min(param.ygrid);
ymax=max(param.ygrid);

xmin = exp(ymin);                          % no borrowing
xmax = 10*exp(ymax);               % guess an upper bound on a,
   check later that do not exceed it

% Declare function space to approximate a'(a,y)
n=param.n;

% Lower and higher bound for the state space (a,y)
smin=xmin;
smax=xmax;

scale=1/2;
fspace=fundef({'spli', nodeunif(n(1),(.01).^scale,(xmax-xmin+.01)
   .^scale).^(1/scale)+xmin-.01,0,3});

grid=funnode(fspace);
s=gridmake(grid); %collection of  states (all a with y1... all a
   with y2... and so on)

```

```
c=funfitxy(fspace,s,s/2);                    %guess that keep constant assets

tic
for it=1:101
    cnew=c;
    x = solve_ca(param,c,fspace,s);
    c=funfitxy(fspace,s,x);

    fprintf('%4i %6.2e\n',[it,norm(c-cnew)]);
    if norm(c-cnew)<1e-5, break, end
end
toc
end
```

**solve_ca.m**

```
function [x] = solve_ca(param,c, fspace, s)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

ns=length(s);

a=.01*ones(ns,1);
b=s;
tol=1e-8; %tolerance level

fa=euler_ca(a,c,fspace,s,param);
fb=euler_ca(b,c,fspace,s,param);

x=zeros(ns,1);

% Start bisection
dx = 0.5*(b - a);

x = a + dx;                          %  start at midpoint
sb=sign(fb);
dx = sb.*dx;                         %  we increase or decrease x depending if f(b) is positive or negative

i=0;
  while any(abs(dx)>tol)
   i=i+1;
    dx = 0.5*dx;
    x = x - sign(euler_ca(x,c,fspace,s,param)).*dx;
```

```
    end


x(fb>=0)=b(fb>=0);
end
```

**euler_ca.m**

```
function fval = euler_ca(x,c,fspace,s,param)

fval=x.^(-param.gamma);
as=size(s,1)/length(param.ygrid);

for i=1:length(param.ygrid)
    cprime=funeval(c,fspace,(1+param.r)*(s-x)+exp(param.ygrid(i)));
%      for j=1:length(param.ygrid)
%          ypp(1+(j-1)*as:j*as,1)=param.yPP(j,i);
%      end
    ypp = param.ws(i);

    fval=fval-param.beta*(1+param.r)*ypp.*cprime.^(-param.gamma);
end

end
```

**markovchain.m**

```
function [con,s] = markovchain(param,c,fspace, T)
%markovchain Generate Markov chain
%   [chain] = markovchain(param,start)
s = ones(T,2);  % Initialize Markov Chain
con = ones(T,1);  % Initialize Markov Chain

s(1,:)= [0, param.ygrid(3)];     % Set starting value

cum_prob = cumsum(param.yPP,2);  % Compute cumulative distribution

% Generate Markov Chain using random numbers uniformly distributed
for t = 2:T
    con(t-1) = funeval(c,fspace,s(t-1,:));
    s(t,1) = (1+param.r)*s(t-1,1)+exp(s(t-1,2))-con(t-1);
    s(t,2) = param.ygrid(find(cum_prob(param.ygrid==s(t-1,2),:)>rand(),1));
end

```

```
end
```

**markovprob.m**

```
function [Gl, TM, CDF, sz] = markovprob(mue, p, s, N, m)
% markovprob - function
%% Arguments:
% mue = intercept of AR(1) process;
% p = slope coeff. of AR(1) process;
% s = std. dev. of residuals in AR(1) process;
% N =  of grid points for the 'z' variable;
% m = Density of the grid for 'z' variable;
%% CODE:
sz = s / ((1-p^2)^(1/2)); % Std. Dev. of z.
zmin = -m * sz + mue/(1-p);
zmax = m * sz + mue/(1-p);
z = linspace(zmin,zmax,N); % Grid Points
%% Transition Matrix:
TM = zeros(N,N); % Transition Matrix
w = z(N) - z(N-1);
for j = 1:N;
TM(j,1) = cdf('norm',(z(1)+w/2-mue-p*z(j))/s,0,1);
TM(j,N) = 1 - cdf('norm',(z(N)-w/2-mue-p*z(j))/s,0,1);
for k = 2:N-1;
TM(j,k) = cdf('norm',(z(k)+w/2-mue-p*z(j))/s,0,1)-cdf('norm',(z(k)-w/2-mue-p*z(j))/s,0,1);
end
end
%% Cumulative Distribution Function:
CDF = cumsum(TM,2);
%% Invariant Distribution:
%% Grids:
Gl = exp(z');
fprintf('If we have a lognormal var. (log(z)) in AR(1) process, \n');
fprintf('To make the interval finer at the lower end and coarser at the upper end.\n');
```