

ECON 6140 - Problem Set # 3

Julien Manuel Neves

April 19, 2018

Problem #1

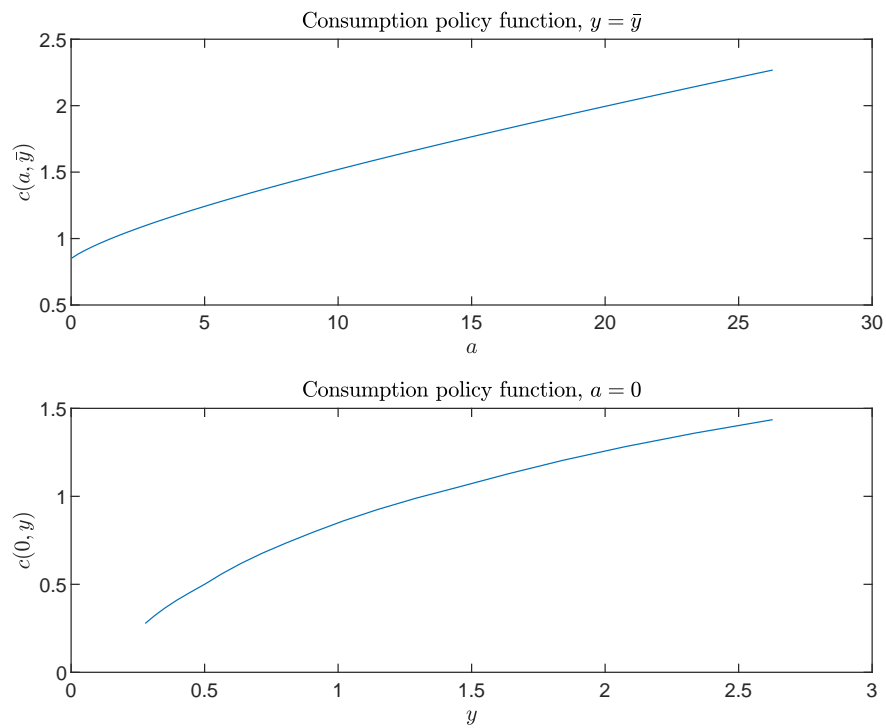


Figure 1: Value function

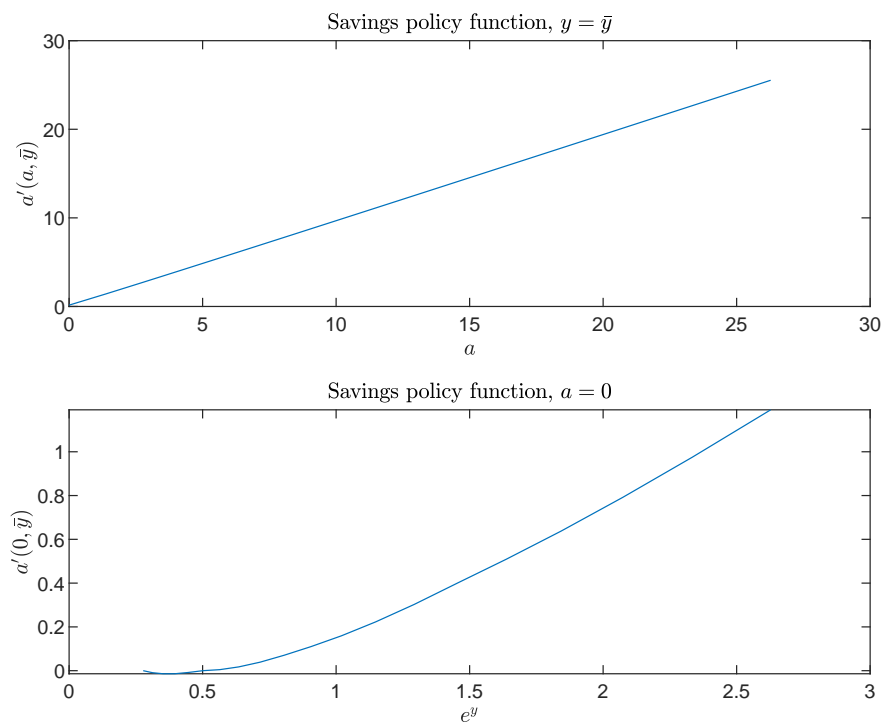


Figure 2: Policy function

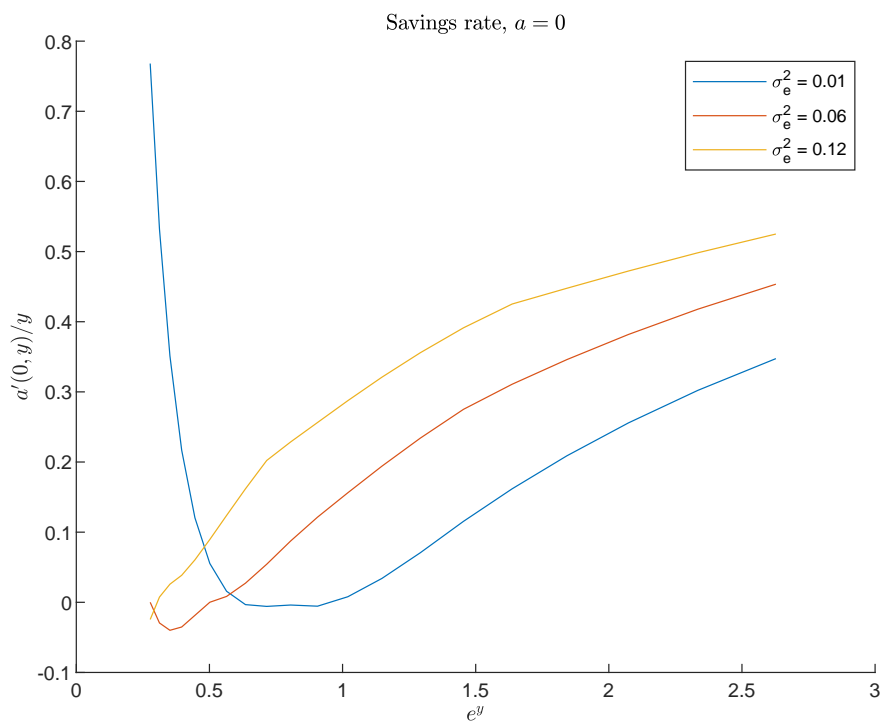


Figure 3: Time path for consumption and capital

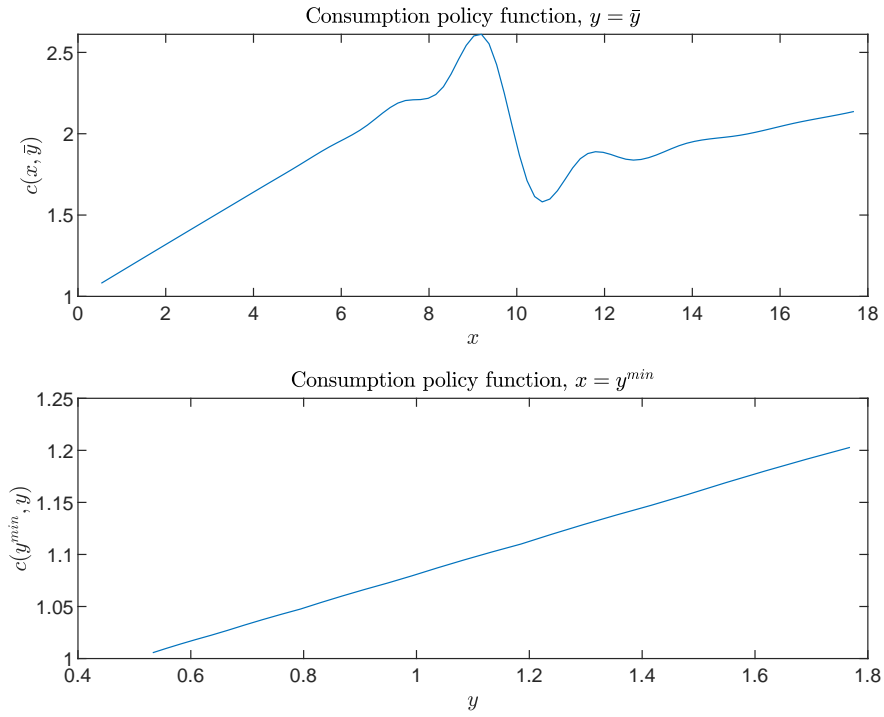


Figure 4: Time path for consumption and capital

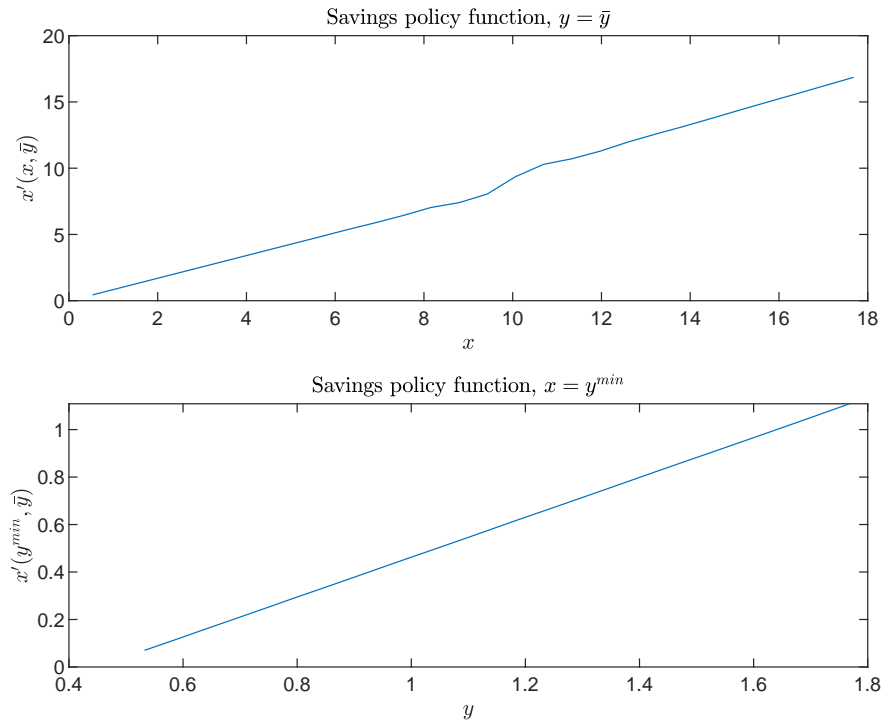


Figure 5: Time path for consumption and capital

Problem #2

Problem #3

Code

0.1 main.m

```
1 clear; clc;
2 %% Problem 1
3 % declare parameters
4 param.ro=0.90;
5 param.se=sqrt(0.06);
6
7 % part a) - b)
8 param.k = 5;
9 [param, ro_ta5, se_ta5] = transition(param, "tauchen");
10 param.k = 10;
11 [param, ro_ta10, se_ta10] = transition(param, "tauchen");
12
13 tab_1b = table([param.ro; ro_ta5; ro_ta10],[param.se; se_ta5;
    se_ta10], 'VariableNames',{ 'rho', 'se' }, 'RowNames',{ 'Model'; 'Five'; 'Ten' })
14
15 % part c)
16 param.k = 5;
17 [param, ro_rw, se_rw] = transition(param, "rouwenhorst");
18
19 tab_1c = table([param.ro; ro_ta5; ro_rw],[param.se; se_ta5; se_rw],
    'VariableNames',{ 'rho', 'se' }, 'RowNames',{ 'Model'; 'tauchen'; 'rouwenhorst' })
20
21 % part d)
22 old_var = param.se^2/(1-param.ro^2);
23 param.ro = 0.98;
24 param.se = old_var*(1-param.ro^2);
25 [param, ro_rw, se_rw] = transition(param, "rouwenhorst");
26
27 tab_1d = table([param.ro; ro_rw],[param.se; se_rw], 'VariableNames',{
    'rho', 'se' }, 'RowNames',{ 'Model'; 'rouwenhorst' })
28
29 %% Problem 2
30 % declare parameters
31 param.beta=0.95;
```

```

32 param.r=0.02;
33 param.ro=0.9;
34 param.se=sqrt(0.06);
35 param.gamma=2;
36 param.amin = 0;
37 param.n = 25;
38 param.k = 5;
39
40 % part a)
41 [param] = transition(param, "rouwenhorst");
42 [param,c,fspace,s,smin,smax] = policy_ip(param);
43
44 close all
45 sfine=gridmake(nodeunif(param.n*2,smin(1),smax(1)),param.ygrid);
46 xfine=funeval(c,fspace,sfine);
47
48 figure(1)
49 subplot(2,1,1)
50 sfine=gridmake(nodeunif(param.n*4,smin(1),smax(1)),0); %ygrid(
    floor(k/2)+2));
51 xfine=funeval(c,fspace,sfine);
52 plot(sfine(:,1),xfine)
53 xlabel({'$a$'},'Interpreter','latex')
54 ylabel({'$c(a,\bar{y})$'},'Interpreter','latex')
55 title({'Consumption policy function, $y=\bar{y}$'},'Interpreter','
    latex')
56 set(gca,'FontSize',8);
57
58 subplot(2,1,2)
59 sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));
60 xfine=funeval(c,fspace,sfine);
61 plot(exp(sfine(:,2)),xfine)
62 xlabel('$y$','Interpreter','latex')
63 ylabel('$c(0,y)$','Interpreter','latex')
64 title({'Consumption policy function, $a=0$'},'Interpreter','latex'
    )
65 set(gca,'FontSize',8);
66 print -depsc fig1.eps
67
68 figure(2)
69 subplot(2,1,1)
70 sfine=gridmake(nodeunif(param.k*4,smin(1),smax(1)),0);
71 xfine=funeval(c,fspace,sfine);
72 plot(sfine(:,1),(1+param.r)*sfine(:,1)+exp(sfine(:,2))-xfine)
73 xlabel('$a$','Interpreter','latex')

```

```

74 ylabel(' $a^{\prime}(a,\bar{y})$ ', 'Interpreter', 'latex')
75 title({'Savings policy function, $y=\bar{y}$'}, 'Interpreter', '
    latex')
76 set(gca, 'FontSize', 8);
77
78 subplot(2,1,2)
79 sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));
80 xfine=funeval(c, fspace, sfine);
81 plot(exp(sfine(:,2)), (1+param.r)*sfine(:,1)+exp(sfine(:,2))-xfine)
82 xlabel('$e^y$', 'Interpreter', 'latex')
83 ylabel('$a^{\prime}(0,\bar{y})$', 'Interpreter', 'latex')
84 title({'Savings policy function, $a=0$'}, 'Interpreter', 'latex')
85 set(gca, 'FontSize', 8);
86 print -depsc fig2.eps
87
88 % part b)
89 gamma = [1,2,5];
90 for i = 1:3
91     param.gamma = gamma(i);
92
93     [param] = transition(param, "rouwenhorst");
94     [param,c,fspace] = policy_ip(param);
95
96     con = markovchain(param,c,fspace, 10000); % Generate Markov
        chain
97     se_c(i) = std(con);
98 end
99
100 tab_2b = table(se_c, 'VariableNames', {'std_c'}, 'RowNames', {'Gamma
    = 1'; 'Gamma = 2'; 'Gamma = 5'})
101
102
103 % part c)
104 param.gamma = 2;
105 se = [0.01,0.06,0.12];
106
107 figure(3)
108 hold on
109 for i = 1:3
110     param.se = sqrt(se(i));
111
112     [param] = transition(param, "rouwenhorst");
113     [param,c,fspace] = policy_ip(param);
114
115     sfine=gridmake(0,nodeunif(param.k*4,smin(2),smax(2)));

```

```

116         xfine=funeval(c, fspace, sfine);
117         plot(exp(sfine(:,2)), 1-xfine./exp(sfine(:,2)))
118     end
119     xlabel('$e^y$', 'Interpreter', 'latex')
120     ylabel('$a^{\prime}(0,y)/y$', 'Interpreter', 'latex')
121     title({'Savings rate, $a=0$'}, 'Interpreter', 'latex')
122     legend('\sigma_e^2 = 0.01', '\sigma_e^2 = 0.06', '\sigma_e^2 = 0.12',
123           )
124     set(gca, 'FontSize', 8);
125     hold off
126     print -depsc fig3.eps
127
128 % part d) - e)
129 % no-borrowing
130 param.gamma = 2;
131 param.se = sqrt(0.06);
132 param.amin = 0;
133
134 [param] = transition(param, "rouwenhorst");
135 [param, c, fspace] = policy_ip(param);
136
137 [con, s] = markovchain(param, c, fspace, 10000);
138 avg_c(1) = mean(con);
139
140 c_t = log(con(2:end))-log(con(1:end-1));
141 e_t = s(2:end, 2)-param.ro*s(1:end-1, 2);
142 sig = cov(c_t, e_t);
143 phi(1) = 1-sig(1,2)/sig(2,2);
144
145 % natural debt limit
146 param.amin = -min(exp(param.ygrid)+.01)/param.r;
147
148 [param] = transition(param, "rouwenhorst");
149 [param, c, fspace] = policy_ip(param);
150
151 [con, s] = markovchain(param, c, fspace, 10000);
152 avg_c(2) = mean(con);
153
154 c_t = log(con(2:end))-log(con(1:end-1));
155 e_t = s(2:end, 2)-param.ro*s(1:end-1, 2);
156 sig = cov(c_t, e_t);
157 phi(2) = 1-sig(1,2)/sig(2,2);
158
159 tab_2d = table(avg_c, phi, 'VariableNames', {'mean_c', 'phi'}, '

```

```

    RowNames',{ 'No borrowing'; 'Natural'})
160
161
162 %% Problem 3
163 % declare parameters
164 param.beta=0.95;
165 param.r=0.02;
166 param.ro=0;
167 param.se=sqrt(0.06);
168 param.gamma=2;
169 param.amin = 0;
170 param.n = 25;
171 param.k = 7;
172
173 % part b)
174 wbar = -param.se^2/2;
175 [x,w] = qwnorm(7,wbar,param.se^2);
176 fprintf('E(y)=%f \n',exp(x)*w)
177
178 % part c)
179 % param.ws = w';
180 % param.ygrid = x;
181 [param] = transition(param, "rouwenhorst");
182 [param,c,fspace,s,smin,smax] = policy_ca(param);
183
184 figure(4)
185 subplot(2,1,1)
186 sfine=gridmake(nodeunif(param.n*4,smin(1),smax(1)),0); %ygrid(
    floor(k/2)+2));
187 xfine=funeval(c,fspace,sfine);
188 plot(sfine(:,1),xfine)
189 xlabel({'$x$'},'Interpreter','latex')
190 ylabel({'$c(x,\bar{y})$'},'Interpreter','latex')
191 title({'Consumption policy function, $y=\bar{y}$'},'Interpreter','
    latex')
192 set(gca,'FontSize',8);
193
194 subplot(2,1,2)
195 sfine=gridmake(smin(1),nodeunif(param.k*4,smin(2),smax(2)));
196 xfine=funeval(c,fspace,sfine);
197 plot(exp(sfine(:,2)),xfine)
198 xlabel({'$y$'},'Interpreter','latex')
199 ylabel({'$c(y^{\min},y)$'},'Interpreter','latex')
200 title({'Consumption policy function, $x=y^{\min}$'},'Interpreter','
    latex')

```



```

201 set(gca, 'FontSize', 8);
202 print -depsc fig4.eps
203
204 figure(5)
205 subplot(2,1,1)
206 sfine=gridmake(nodeunif(param.k*4,smin(1),smax(1)),0);
207 xfine=funeval(c, fspace, sfine);
208 plot(sfine(:,1), (1+param.r)*(sfine(:,1)-xfine)+exp(sfine(:,2)))
209 xlabel('$x$', 'Interpreter', 'latex')
210 ylabel('$x^{\backslash prime}(x, \backslash bar{y})$', 'Interpreter', 'latex')
211 title({'Savings policy function, $y=\backslash bar{y}$'}, 'Interpreter', '
    latex')
212 set(gca, 'FontSize', 8);
213
214 subplot(2,1,2)
215 sfine=gridmake(smin(1), nodeunif(param.k*4,smin(2),smax(2))),0);
216 xfine=funeval(c, fspace, sfine);
217 plot(exp(sfine(:,2)), (1+param.r)*sfine(:,1)+exp(sfine(:,2))-xfine)
218 xlabel('$y$', 'Interpreter', 'latex')
219 ylabel('$x^{\backslash prime}(y^{\backslash min}, \backslash bar{y})$', 'Interpreter', 'latex')
220 title({'Savings policy function, $x=y^{\backslash min}$'}, 'Interpreter', '
    latex')
221 set(gca, 'FontSize', 8);
222 print -depsc fig5.eps

```

0.2 tauchen.m

```

1 function [state_grid, prob] = tauchen(rho, sig_eps, ns, dev)
2 %markovchain Discretize AR(1) process
3 % [prob, cum_prob, state_grid] = tauchen(ns, rho, sig_y, dev)
4 returns
5 % prob, transition matrix,
6 % cum_prob, cumulative distribution
7 % state_grid, value of states
8
9
10
11 sig_y = sig_eps/sqrt(1-rho^2); % Compute std of epsilon
12
13
14 state_grid = linspace(-dev*sig_y, dev*sig_y, ns)'; % Set state
    grid
15 d = mean(diff(state_grid));
16 % Compute  $y^j_{t+1} - \rho y^i_t$ 
17 state_change = repmat(state_grid, 1, ns)' - rho*repmat(state_grid
    , 1, ns);
18
19

```

```

16
17 prob = zeros(ns,ns);    % Initialize transition matrix
18 % Compute transition matrix according to Tauchen (1986)
19 prob(:,1) = cdf('norm',state_change(:,1)+d/2,0,sig_eps);
20 prob(:,end) = 1- cdf('norm',state_change(:,end) - d/2,0,sig_eps);
21 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);
22
23 state_grid = state_grid';
24 end

```

0.3 rouwenhorst.m

```

1 %rouwenhorst.m
2 %
3 %[zgrid, P] = rouwenhorst(rho, sigma_eps, n)
4 %
5 % rho is the 1st order autocorrelation
6 % sigma_eps is the standard deviation of the error term
7 % n is the number of points in the discrete approximation
8 %
9 function [zgrid, P] = rouwenhorst(rho, sigma_eps, n)
10
11 mu_eps = 0;
12
13 q = (rho+1)/2;
14 nu = ((n-1)/(1-rho^2))^(1/2) * sigma_eps;
15
16 P = [q 1-q; 1-q q];
17
18
19 for i=2:n-1
20     P = q*[P zeros(i,1); zeros(1,i+1)] + (1-q)*[zeros(i,1) P; zeros
        (1,i+1)] + ...
21     (1-q)*[zeros(1,i+1); P zeros(i,1)] + q*[zeros(1,i+1); zeros
        (i,1) P];
22     P(2:i,:) = P(2:i,:)/2;
23 end
24
25 zgrid = linspace(mu_eps/(1-rho)-nu, mu_eps/(1-rho)+nu, n);

```

0.4 transition.m

```

1 function [param, ro_tilde, se_tilde] = transition(param, method)
2 %UNTITLED2 Summary of this function goes here
3 % Detailed explanation goes here

```

```

4
5 var_w= param.se^2/(1-param.ro^2);
6 wbar = -var_w*(1-param.ro)/2;
7 % nodes for the distribution of income shocks
8
9 if method=="tauchen"
10     [e,w] = tauchen(param.ro,param.se,param.k,3);
11 elseif method=="rouwenhorst"
12     [e,w] = rouwenhorst(param.ro,param.se,param.k); %
13     Rouwenhorst method
14 end
15 param.yPP=w;
16 param.ygrid=e';
17
18 [~,D,V]=eig(w);
19 ws = V(:,1);
20 ws = ws/sum(ws);
21
22 param.ws = ws;
23 param.ygrid = param.ygrid + wbar/(1-param.ro);
24
25 ro_tilde = e*(w*e')/(e*e');
26 se_tilde = sqrt(e.^2*ws*(1-ro_tilde^2));
27 end

```

0.5 policy_ip.m

```

1 function [param,c,fspace,s,smin,smax] = policy_ip(param)
2 % Bounds for state space
3 ymin=min(param.ygrid);
4 ymax=max(param.ygrid);
5
6 amin = param.amin; % no borrowing
7 amax = 10*exp(ymax); % guess an upper bound on a,
8 % check later that do not exceed it
9
10 % Declare function space to approximate a'(a,y)
11 n=[param.n,param.k];
12
13 % Lower and higher bound for the state space (a,y)
14 smin=[amin,ymin];
15 smax=[amax,ymax];
16
17 scale=1/2;

```

```

17 fspace=fundef({ 'spli', nodeunif(n(1),(smin(1)-amin+.01).^scale,(
    smax(1)-amin+.01).^scale).(1/scale)+amin-.01,0,3},...
18     {'spli', param.ygrid,0,1});
19
20 grid=funnode(fspace);
21 s=gridmake(grid); %collection of states (all a with y1... all a
    with y2... and so on)
22
23 c=funfitxy(fspace,s,param.r/(1+param.r)*s(:,1)+exp(s(:,2)));
    %guess that keep constant assets
24
25 tic
26 for it=1:101
27     cnew=c;
28     x = solve_ip(param,c,fspace,s,amin);
29     c=funfitxy(fspace,s,x);
30
31     fprintf('%4i %6.2e\n',[it,norm(c-cnew)]);
32     if norm(c-cnew)<1e-7, break, end
33 end
34 toc
35 end

```

0.6 solve_ip.m

```

1 function [x] = solve_ip(param,c,fspace,s,amin)
2 %UNTITLED3 Summary of this function goes here
3 % Detailed explanation goes here
4
5 ns=length(s);
6
7 a=.01*ones(ns,1);
8 b=(1+param.r)*s(:,1)+exp(s(:,2))-amin;
9 tol=1e-8; %tolerance level
10
11 fa=euler_ip(a,c,fspace,s,param);
12 fb=euler_ip(b,c,fspace,s,param);
13
14 x=zeros(ns,1);
15
16 % Start bisection
17 dx = 0.5*(b - a);
18
19 x = a + dx; % start at midpoint
20 sb=sign(fb);

```

```

21 dx = sb.*dx; % we increase or decrease x
    depending if f(b) is positive or negative
22
23 i=0;
24 while any(abs(dx)>tol)
25     i=i+1;
26     dx = 0.5*dx;
27     x = x - sign(euler_ip(x,c,fspace,s,param)).*dx;
28 end
29
30
31 x(fb>=0)=b(fb>=0);
32 end

```

0.7 euler_ip.m

```

1 function fval = euler_ip(x,c,fspace,s,param)
2
3 ns=size(s,1);
4 a=s(:,1);
5 y=exp(s(:,2));
6 aprime=(1+param.r)*a+y-x;
7 fval=x.^(-param.gamma);
8 as=ns/length(param.ygrid);
9
10 for i=1:length(param.ygrid)
11     cprime=funeval(c,fspace,[aprime,param.ygrid(i)*ones(ns,1)]);
12     for j=1:length(param.ygrid)
13         ypp(1+(j-1)*as:j*as,1)=param.yPP(j,i);
14     end
15     fval=fval-param.beta*(1+param.r)*ypp.*cprime.^(-param.gamma);
16 end

```

0.8 policy_ca.m

```

1 function [param,c,fspace,s,smin,smax] = policy_ca(param)
2 % Bounds for state space
3 ymin=min(param.ygrid);
4 ymax=max(param.ygrid);
5
6 xmin = exp(ymin); % no borrowing
7 xmax = 10*exp(ymax); % guess an upper bound on a,
    check later that do not exceed it
8
9 % Declare function space to approximate a'(a,y)
10 n=[param.n,param.k];

```

```

11
12 % Lower and higher bound for the state space (a,y)
13 smin=[xmin,ymin];
14 smax=[xmax,ymax];
15
16 scale=1/2;
17 fspace=fundef({ 'spli', nodeunif(n(1),(smin(1)-xmin+.01).^scale,(
    smax(1)-xmin+.01).^scale).^(1/scale)+xmin-.01,0,3},...
18     {'spli', param.ygrid,0,1});
19
20 grid=funnode(fspace);
21 s=gridmake(grid); %collection of states (all a with y1... all a
    with y2... and so on)
22
23 c=funfitxy(fspace,s,param.r/(1+param.r)*s(:,1)+exp(s(:,2)));
    %guess that keep constant assets
24
25 tic
26 for it=1:101
27     cnew=c;
28     x = solve_ca(param,c,fspace,s);
29     c=funfitxy(fspace,s,x);
30
31     fprintf('%4i %6.2e\n',[it,norm(c-cnew)]);
32     if norm(c-cnew)<1e-5, break, end
33 end
34 toc
35 end

```

0.9 solve_ca.m

```

1 function [x] = solve_ca(param,c,fspace,s)
2 %UNTITLED3 Summary of this function goes here
3 % Detailed explanation goes here
4
5 ns=length(s);
6
7 a=.01*ones(ns,1);
8 b=s(:,1)+exp(s(:,2))/(1+param.r);
9 tol=1e-8; %tolerance level
10
11 fa=euler_ca(a,c,fspace,s,param);
12 fb=euler_ca(b,c,fspace,s,param);
13
14 x=zeros(ns,1);

```

```

15
16 % Start bisection
17 dx = 0.5*(b - a);
18
19 x = a + dx; % start at midpoint
20 sb=sign(fb);
21 dx = sb.*dx; % we increase or decrease x
    depending if f(b) is positive or negative
22
23 i=0;
24 while any(abs(dx)>tol)
25     i=i+1;
26     dx = 0.5*dx;
27     x = x - sign(euler_ca(x,c,fspace,s,param)).*dx;
28 end
29
30
31 x(fb>=0)=b(fb>=0);
32 end

```

0.10 euler_ca.m

```

1 function fval = euler_ca(x,c,fspace,s,param)
2
3 ns=size(s,1);
4 xprime = (1+param.r)*(s(:,1)-x)+exp(s(:,2));
5 fval=x.^(-param.gamma);
6
7 for i=1:length(param.ygrid)
8
9     cprime=funeval(c,fspace,[xprime,param.ygrid(i)*ones(ns,1)]);
10    as=ns/length(param.ygrid);
11    for j=1:length(param.ygrid)
12        ypp(1+(j-1)*as:j*as,1)=param.yPP(j,i);
13    end
14    % ypp = param.ws(i);
15    fval=fval-param.beta*(1+param.r)*ypp.*cprime.^(-param.gamma);
16 end
17
18 end

```

0.11 markovchain.m

```

1 function [con,s] = markovchain(param,c,fspace,T)
2 %markovchain Generate Markov chain
3 % [chain] = markovchain(param,start)

```

```

4 s = ones(T,2); % Initialize Markov Chain
5 con = ones(T,1); % Initialize Markov Chain
6
7 s(1,:) = [0, param.ygrid(3)]; % Set starting value
8
9 cum_prob = cumsum(param.yPP,2); % Compute cumulative distribution
10
11 % Generate Markov Chain using random numbers uniformly distributed
12 for t = 2:T
13     con(t-1) = funeval(c, fspace, s(t-1,:));
14     s(t,1) = (1+param.r)*s(t-1,1)+exp(s(t-1,2))-con(t-1);
15     s(t,2) = param.ygrid(find(cum_prob(param.ygrid==s(t-1,2),:)>
        rand(),1));
16 end
17
18 end

```

0.12 markovprob.m

```

1 function [Gl, TM, CDF, sz] = markovprob(mue, p, s, N, m)
2 % markovprob - function
3 %% Arguments:
4 % mue = intercept of AR(1) process;
5 % p = slope coeff. of AR(1) process;
6 % s = std. dev. of residuals in AR(1) process;
7 % N = of grid points for the 'z' variable;
8 % m = Density of the grid for 'z' variable;
9 %% CODE:
10 sz = s / ((1-p^2)^(1/2)); % Std. Dev. of z.
11 zmin = -m * sz + mue/(1-p);
12 zmax = m * sz + mue/(1-p);
13 z = linspace(zmin,zmax,N); % Grid Points
14 %% Transition Matrix:
15 TM = zeros(N,N); % Transition Matrix
16 w = z(N) - z(N-1);
17 for j = 1:N;
18     TM(j,1) = cdf('norm',(z(1)+w/2-mue-p*z(j))/s,0,1);
19     TM(j,N) = 1 - cdf('norm',(z(N)-w/2-mue-p*z(j))/s,0,1);
20     for k = 2:N-1;
21         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);
22     end
23 end
24 %% Cumulative Distribution Function:
25 CDF = cumsum(TM,2);

```



```

26 %% Invariant Distribution:
27 %% Grids:
28 G1 = exp(z');
29 fprintf('If we have a lognormal var. (log(z)) in AR(1) process, \n
        ');
30 fprintf('To make the interval finer at the lower end and coarser
        at the upper end.\n');

```