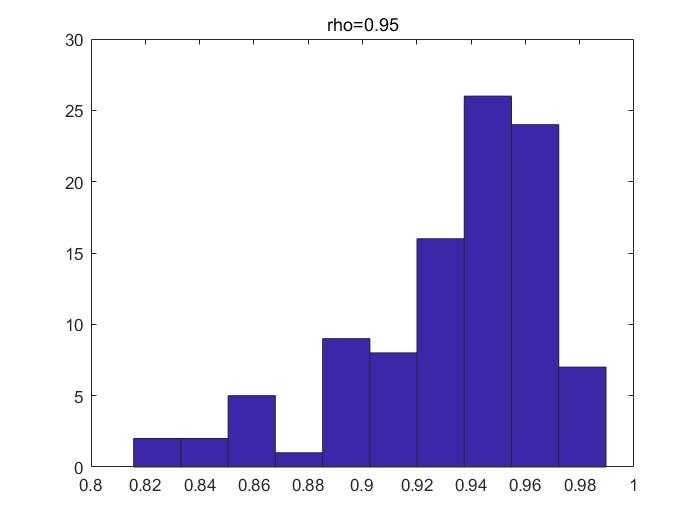
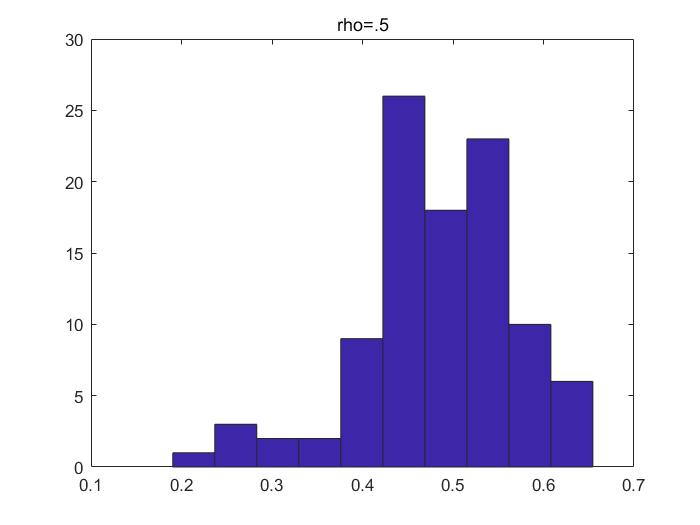
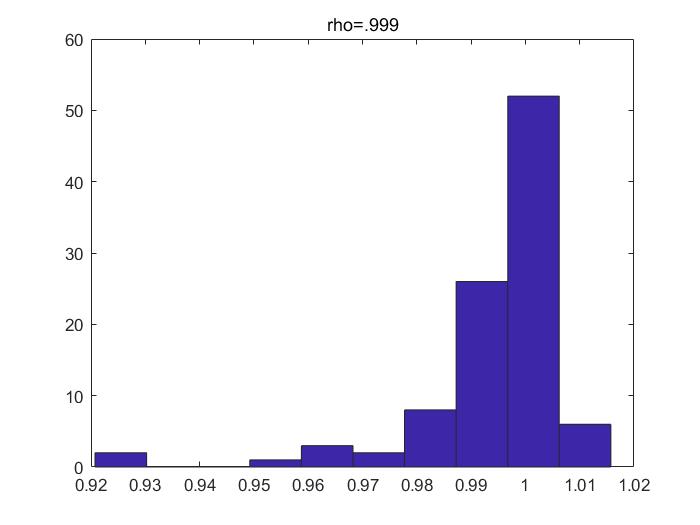
Simulating an AR(1):

The histogram of rho = 0.95, 0.50, 0.999 were shown in the following respectively:







From the above graphs, all the estimators had a long-left-tail distribution. Thus, the OLS method is biased in small samples.

The Coding for this problem (plot commands are omitted):

%% (b) AR(1) generation input is y0, T, alpha, rho;

function Y=AR1T(T,alpha,rho)

% initialize Y

Y=zeros(T,1);

for i=1:T

if i==1

Y(i)=alpha+randn;

else

Y(i)=rho\*Y(i-1)+alpha+randn;

end

end

%% (a) OLS Function Practise Input are Y and X;

function [betahat, cvar, se] = olsl(Y, X)

% 1. data credibility check

[ys1,ys2]=size(Y);

[xs1,xs2]=size(X);

if ys1~=xs1

error('Y and X observations" # not equal!' );

elseif ys2>1

error('Y is not a column vector');

elseif xs1<xs2

error('# of observations is less than # of variables');

end

% 2. estimation of betahat, cvar and standard error of estimate

betahat=(X'\*X)^(-1)\*X'\*Y;

Px=X\*(X'\*X)^(-1)\*X';

Mx=eye(xs1)-Px;

e=Mx\*Y;

deltahat=e'\*e/(xs1-xs2);

cvar=deltahat\*(X'\*X)^(-1);

se=diag(cvar);

end

%% Problem C

rhohat=zeros(100,3);

T=1100;

rho=[.95,.5,.999];

alpha=0;

for j=1:3

for i=1:100

Y0=AR1T(T,alpha,rho(j));

Y0Last=Y0(1001:1100);

X0Last=Y0(1000:1099);

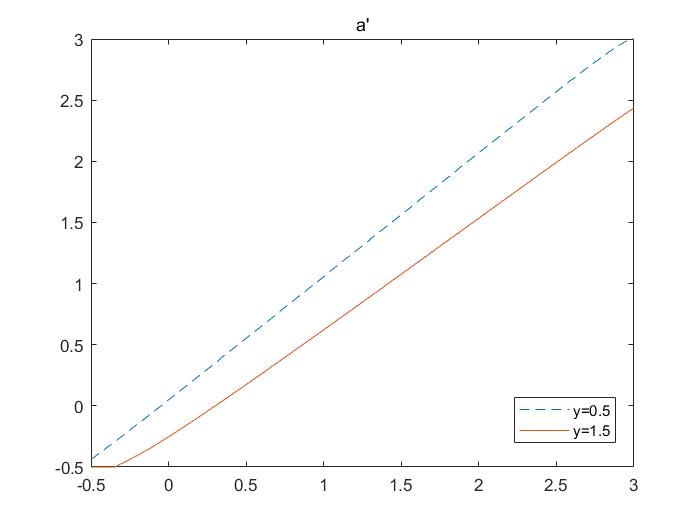
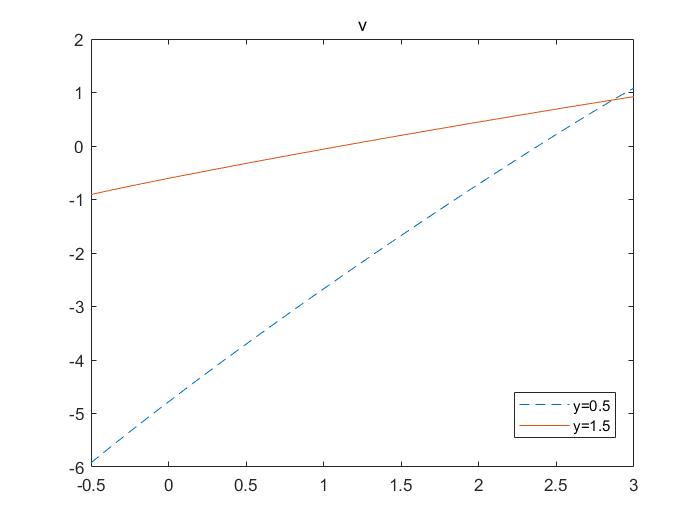
[rhohat(i,j),tmp1,tmp2]=olsl(Y0Last,X0Last);

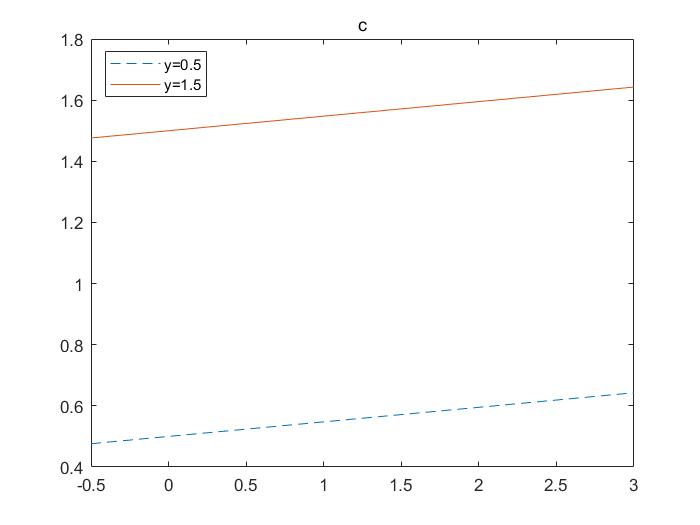
end

end

Infinite-horizon consumption and saving with uncertainty

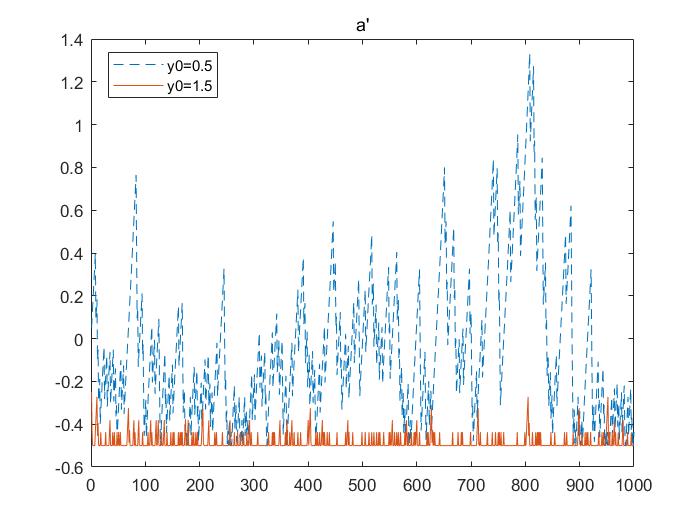
1. The graphs of V, a’ and c in the solution were shown in the following, where and :

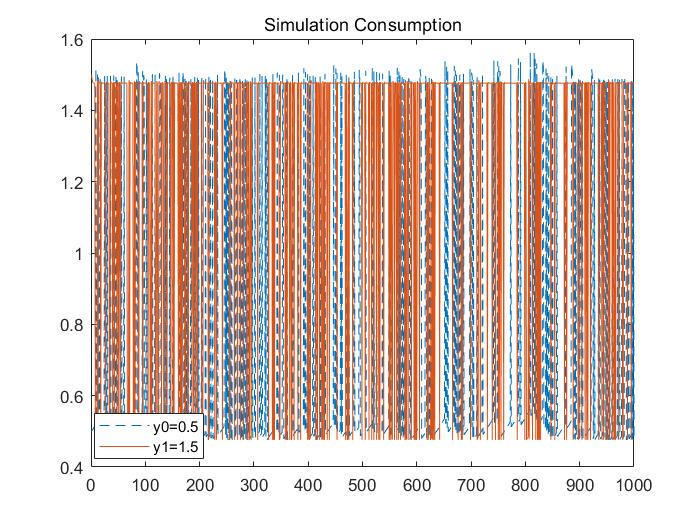




The a’ and c seemed in the right shapes, while the v’s shape was dubious to be right.

1. a0=0 was chosen as the initial condition the graphs of simulation results were presented in the following:





1. When increases, v will increase and y=0.5 will increase more than y=1.5, a’ will increase and y=0.5 will increase more than y=1.5, c will not change, vice versa.

When r increases, v, a’ and c will increase, while v and a’ y=0.5 will increase more than y=1.5 and c in y=0.5 and y=1.5 will increase at the same amount.

When y process has more volatile, meaning transferring [0.5, 1.5] to [0.3, 1.7], v will decrease and more in y=0.3 especially in the a’s negative territory. a’ will be more sensitive to a0, as the slope of the line become steeper. c y=0.3 will decrease in y=0.3 and increase in y=1.7, while both paths indicate more smooth shapes.

The Code is as the following:

%% Infinite-horizon consumption and saving with uncertainty

% Initialization of params in the model

r=.05; % return of saving

beta=.95; % discount rate

% Approximation of y by a Markov process given in the helpful note

PI=[.8, .2;.8, .2]; % transition matrix

y=[.5 1.5]; % two suggested states

% Construction of the a's grid

nbg=1000;% # of a's points in the grid

nbs=length(y); % # of y's states

crit=1; % Initialization of the convergence criterion

epsi=1e-6; % Precision of convergence

amin=-.5; % Lower bound for gross saving returns

amax=3; % Upper bound for gross saving returns

agrid=linspace(amin,amax,nbg)'; % The a's grid

% Initialization of variables in the iteration

c=zeros(nbg,nbs);

util=zeros(nbg,nbs);

v=zeros(nbg,nbs);

Tv=zeros(nbg,nbs);

dr=zeros(nbg,nbs);

iter=0;

Maxiter=500;

while and(crit>epsi,iter<Maxiter)

for i=1:nbg

for j=1:nbs

tempc=agrid(i)+y(j)-agrid./(1+r);

neg=find(tempc<=0);

tempc(neg)=NaN;

util(:,j)=log(tempc);

util(neg,j)=-1e12;

end

[Tv(i,:),dr(i,:)]=max(util+beta\*(v\*PI));

end

crit=max(max(abs(Tv-v)));

v=Tv;

iter=iter+1;

end

apolicy=agrid(dr);

for j=1:nbs

c(:,j)=agrid+y(j)\*ones(size(agrid))-agrid./(1+r);

end

function chain=markov\_gen(y0)

transition\_probabilities = [0.8 0.2;0.2 0.8]; chain\_length = 1000;

chain = zeros(chain\_length,1);

chain(1)=y0;

posval=[0.5,1.5];

for i=2:chain\_length

this\_step\_distribution = transition\_probabilities(find(posval==y0),:);

cumulative\_distribution = cumsum(this\_step\_distribution);

r = rand();

chain(i) = posval(find(cumulative\_distribution>r,1));

end

end

%% 2Q in 2P simulation

% Initialization of a0, c0 and total time;

a0=0;

n=1000;

Sapolicy=zeros(n,2);

Sc=zeros(n,2);

% Generation of the optimal solution

cnsUncertaintyValueFunctionIteration;

% Simulation of the y's process:

posval=[0.5 1.5];

y=[markov\_gen(0.5), markov\_gen(1.5)];

idx\_track=zeros(n,2);

for i=1:n

if i==1

[~,idx]=min(abs(agrid));

idx\_track(i,:)=[idx,idx];

Sapolicy(i,:)=[apolicy(idx,find(posval==y(i,1))),apolicy(idx,find(posval==y(i,2)))];

Sc(i,:)=[c(idx,find(posval==y(i,1))),c(idx,find(posval==y(i,2)))];

else

[~,idx1]=min(abs(Sapolicy(i-1,1)-agrid));

[~,idx2]=min(abs(Sapolicy(i-1,2)-agrid));

idx\_track(i,:)=[idx1,idx2];

Sapolicy(i,:)=[apolicy(idx1,find(posval==y(i,1))),apolicy(idx2,find(posval==y(i,2)))];

Sc(i,:)=[c(idx1,find(posval==y(i,1))),c(idx2,find(posval==y(i,2)))];

end

end