mcs_mean.m

% Monte Carlo Simulation of gradient learning, Simple Gradient without ranking utility
% Hui Miao & Zheng Xiaqian, CMPS272 Project

mean1=0.0014;
std1=0.02;
R=0.001;
C=0.05;
time=10000;
agent=100;
adjust=0.05;

beta=zeros(1,agent);
x=zeros(time,agent);
wealth=zeros(time,agent);
x(1,1:agent) = rand(1,agent);  %uniform distributed initially
wealth(1,1:agent) = 1;  %initial endowment

for i=2:time,
    r = mean1+std1*randn(1);
    %calculate gradient update
    for j=1:agent,
        wealth(i,j) = (r-R)*x(i-1,j) + (R+1)*wealth(i-1,j);
    end

    tmp=mean(wealth(i,1:agent));
    for j=1:agent,
        wealth(i,j)=wealth(i,j)/tmp;  %normalize wealth
    end

    for j=1:agent,
        if(wealth(i,j)<=0),  %lose all money
            x(i,j)=0;
        else
            x(i,j)=min(wealth(i,j),x(i-1,j)*(1+adjust*(r-R)*tmp/wealth(i,j)));  % myopia learning, one period
        end
    end

end

for j=1:agent,
    if(wealth(time,j)>0)
        beta(j)= tmp*x(time-1,j)/wealth(time,j);
    else
        beta(j)=0;
    end
end;

plot(beta);

% this program shows that the pdf of beta converges towards 0.65,
% c=0.1, beta will above 0.7, c=0.01, beta will be below 0.6, good results
% Monte Carlo Simulation of gradient learning, Gradient learning with ranking utility
% Hui Miao & Zheng Xiaqian, CMPS272 Project

mean1=0.14;
std1=0.2;
R=0.1;
C=5;
time=10000;
agent=100;
adjust=0.05;

beta=zeros(1,agent);
x=zeros(time,agent);
wealth=zeros(time,agent);
x(1,1:agent) = rand(1,agent); %uniform distributed initially
wealth(1,1:agent) = 1; %initial endowment

for i=2:time,
    betas=zeros(1,agent);
    % calculate beta and wealth
    r = mean1+std1*randn(1); %normal distribution
    for j=1:agent,
        wealth(i,j) = (r-R)*x(i-1,j) + (R+1)*wealth(i-1,j); %update wealth
        if(wealth(i-1,j)>0)
            beta(j)=x(i-1,j)/wealth(i-1,j);
        else
            beta(j)=0;
        end
    end

    tmp=mean(wealth(i,1:agent));
    for j=1:agent,
        wealth(i,j)=wealth(i,j)/tmp; %normalize wealth
    end

% calculate rank
    rank=zeros(1,agent);
    betas=sort(beta);
    for j=1:agent
        for m=1:agent,
            if beta(j) >= betas(m)
                rank(j)=rank(j)+1;
            end
        end
    end

% calculate the rank-dependent gradient
    for j=1:agent,
        if(wealth(i,j)<=0), %lose all money
            x(i,j)=0;
        else % greedy method, local learning, could add more complicated greedy search here
            %
        end
end
if(r>=R)
    x(i,j)=max(0,min(wealth(i,j),x(i-1,j)*(1+adjust*(rank(j)/agent+C*(r-R)*tmp/wealth(i, j))))); % myopia learning, one period
else
    x(i,j)=max(0,min(wealth(i,j),x(i-1,j)*(1+adjust*(-rank(j)/agent+C*(r-R)*tmp/wealth(i, j))))); % myopia learning, one period
end
end
end
plot(beta);