Homework 5.(b)

Modeling Complex Systems, Javier Lobato

Due date: Thursday, May 3, 2018

Section 1

The modifications done to the code are explained with the comments along the code. The way of storing the RBN with the variable length depending on the Poisson distribution is cell arrays. A cell array with length N will refer to each individual of the RBN. Inside each element $i \in N$ there will be a vector of variable length. The length will be given by the Poisson distribution, and it will contain the number of the cells to which they refer (given by a random permutation).

The way that the system is updated is similar to before, but using cell arrays instead plain arrays. For each timestep, each element in the RBN will be analyzed, looking for the position of the individuals which it depends on. Getting the position of the index that returns that value in the list, and substituting it, the new state of that individual has been computed.

Section 2

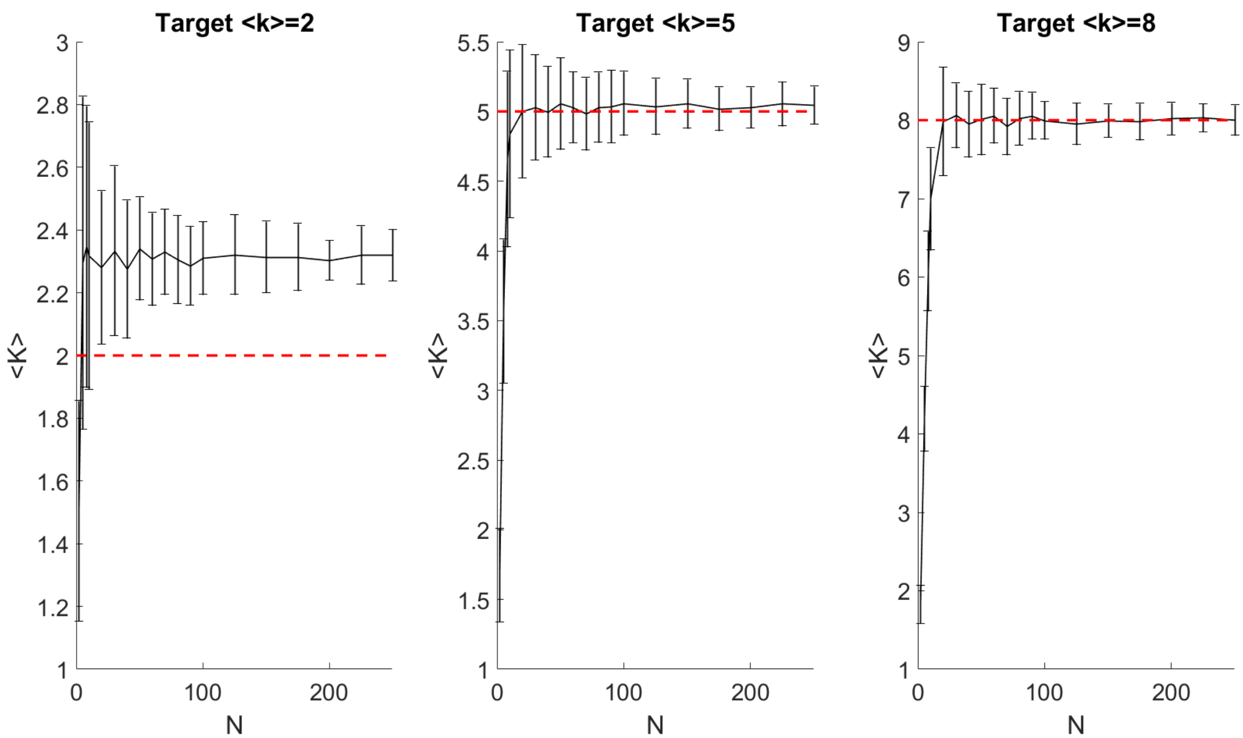
For section 2, the RBN of *Homework 5.a* was implemented as specified to use the different modified functions. When executing just one timestep for each possible input, the state matrix is obtained (as shown in the image below). In order to fully test the code, the same system was evolved beginning with the first element of the basin of attraction that ends in the periodic 2 attractors (`0 1 0`) for 7 timesteps.

```
outputMat =

     0     0     0
     0     0     0
     1     1     0
     1     0     0
     0     1     1
     0     1     1
     1     0     1
     1     1     1

>> evolveRBN(C,rules,[0,1,0],7)

ans =

     0     1     0
     1     1     0
     1     0     1
     0     1     1
     1     0     0
     0     1     1
     1     0     0
     0     1     1

fx >>
```

Execution of the section 2 and the periodic attractor of the system

Section 3

Running the same simulation with a target $< k >$ may give results that differ from the expected value. In order to test that, the same setup was run various times, averaging the results. The value of the probability does not really matters, because only the individuals are being analyzed and not so the rules.

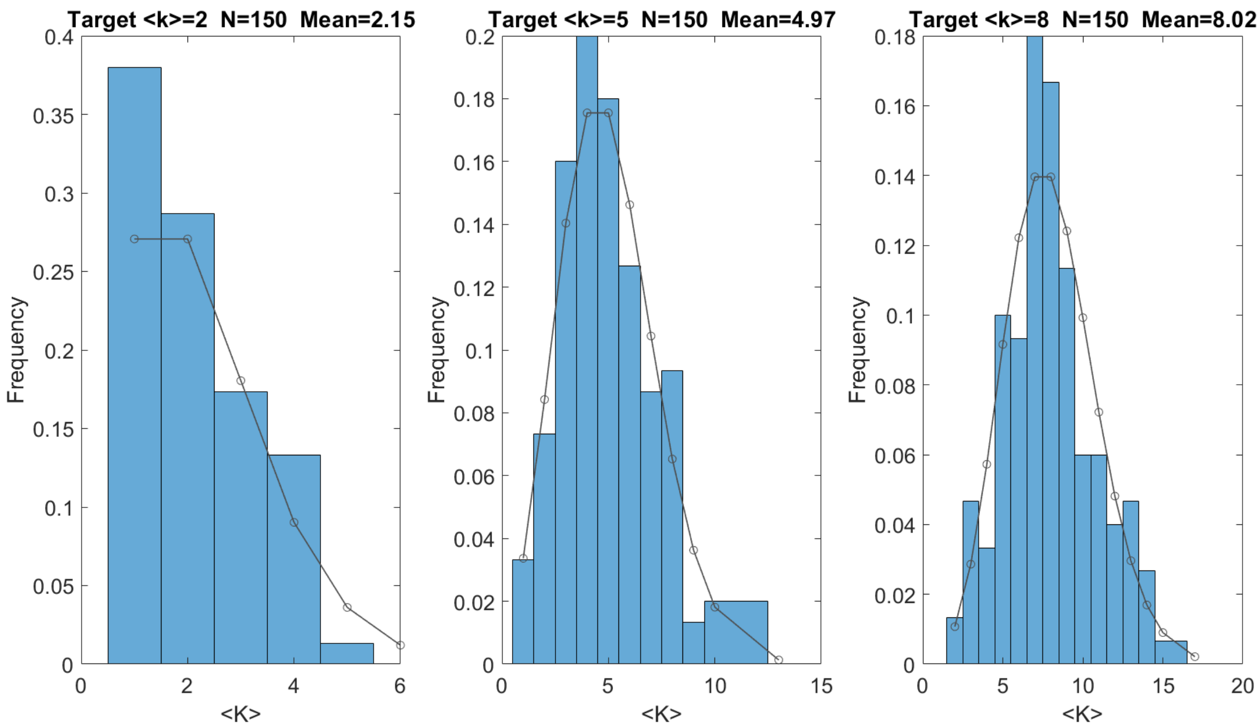


Real $< k >$ value for differents N and target $< K >$

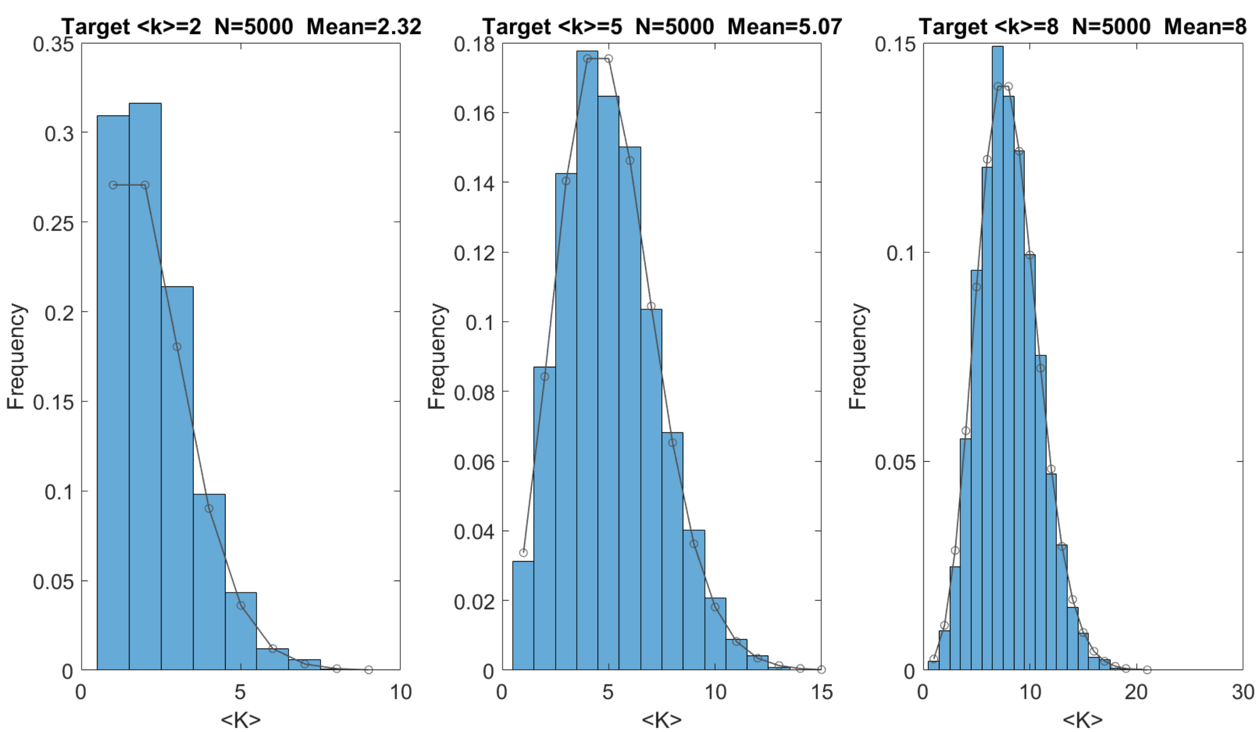
As it can be seen, once the value of N goes over a threshold that increases with K, the value of the mean is close enough to the target $< k >$ value. However, the standard deviations are too big until a high number of N is achieved.

Section 4

To get the actual value of $< k >$, the length of each one of the arrays for each element in the RBN is computed. When plotting it in a histogram, a Poisson distribution must be observed. If $N = 150$ is chosen, the value of the means are close to the expected value. However, the shape of the distribution is not optimal. If a bigger number of individuals is used (e.g. $N = 500$), both the shape and the mean value is closer to the expected ones. The true value of the Poisson distribution for each $< k >$ has been also computed, showing this way how close the distribution is to the expected value.

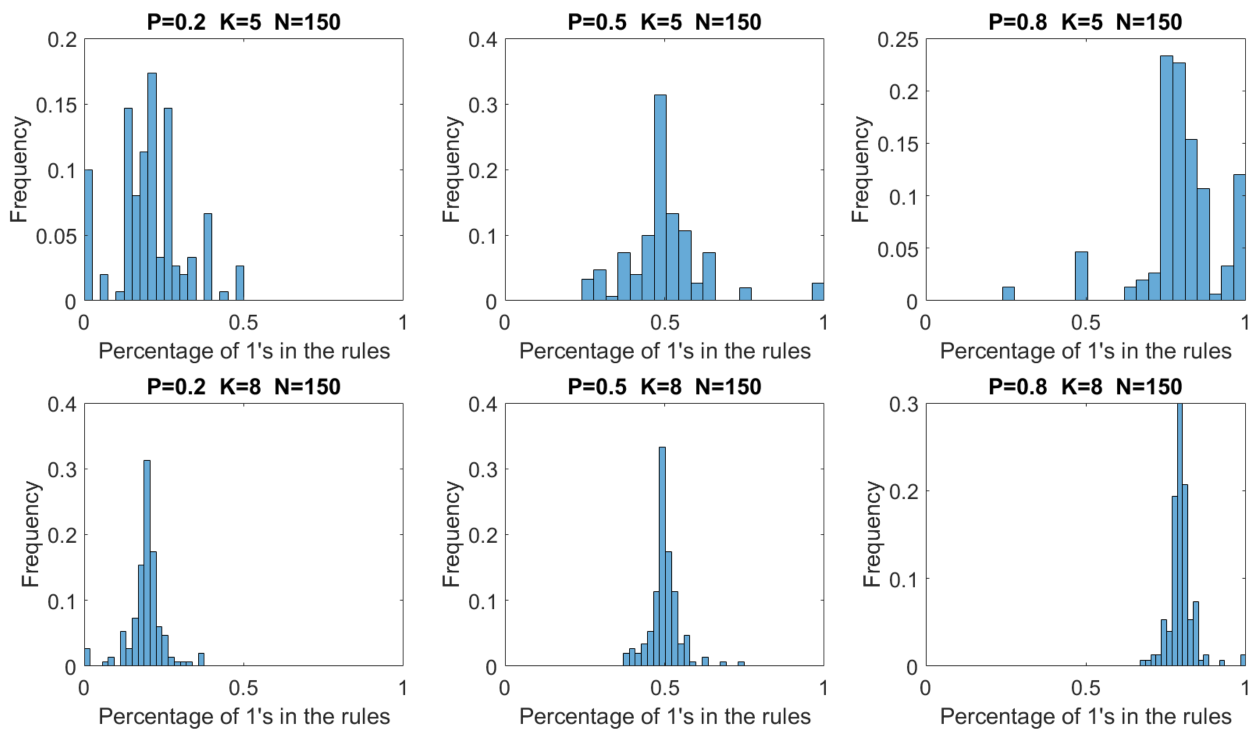


Different target $< k >$ for a RBN of $N = 150$



Different target $< k >$ for a RBN of $N = 5000$

To analyze how the value of p (probability of having 1's in the output) change the distribution of the rules (for different values of the target $< k >$) the analysis was done to the rules instead of doing it to the individuals. To do it, the number of ones in each rule was counted, dividing it by the length of the rule. The histograms of those values for different values of P and $< k >$ are shown below:



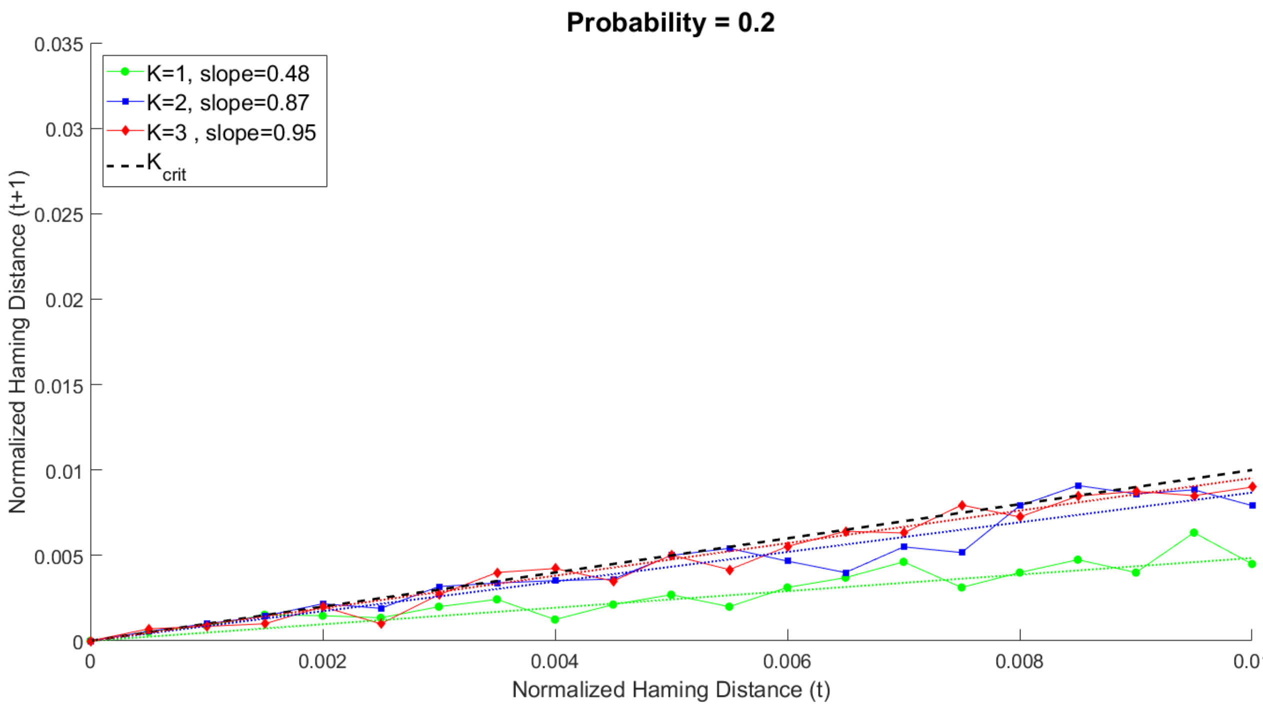
Variation of p and k and the effect on the rules

Section 5

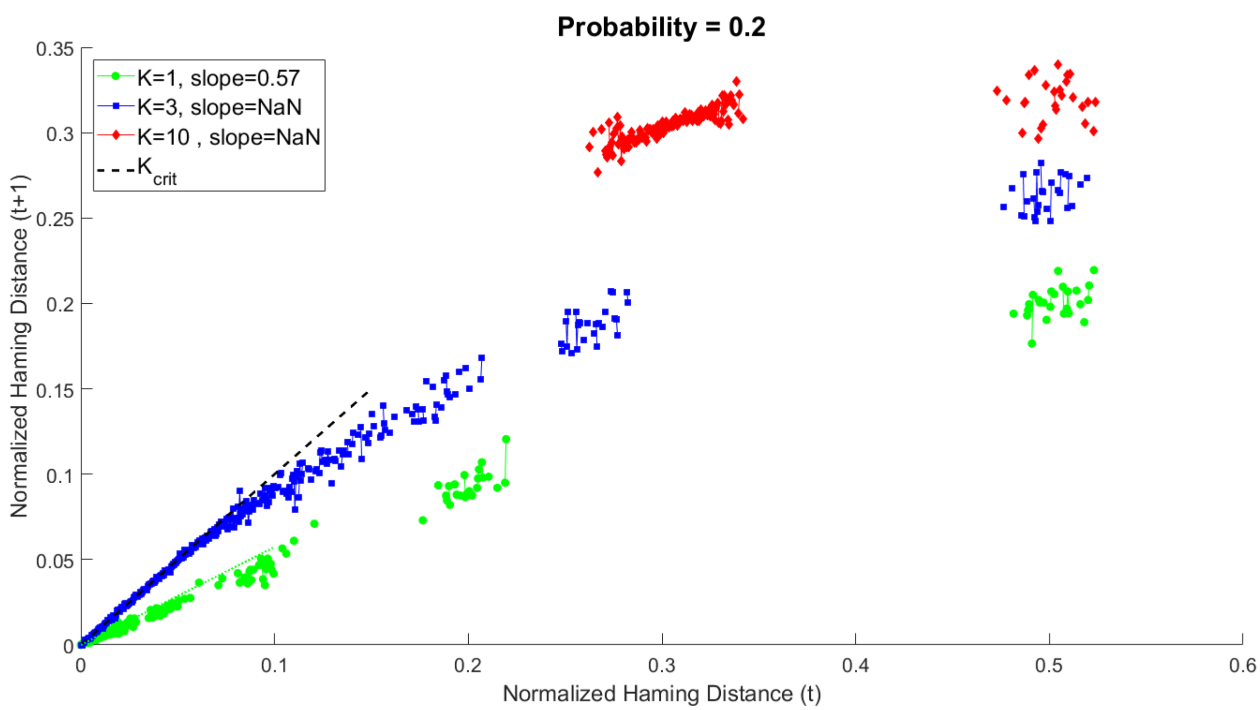
The Derrida plots of this systems are shown below. The dynamics of the system are harder to determine than for the previous homework assignment.

- $p = 0.2$: the slopes indicate that for $k < 3$ are in the stable regime, given that the slope is smaller than 1. However, if the plot is shown not only in the region close to the origin, it can be seen that higher k values give a critical regime - given that they are above the line with a slope of 1.
- $p = 0.5$: in the region close to the origin, the points of $K = 2$ don't appear, so it may be inferred that it is already at criticality. However, if the limits of the plot are removed, it can be seen that it is still in the stable zone.
- $p = 0.8$: the results of this case are close to the ones obtained with $p = 0.2$.

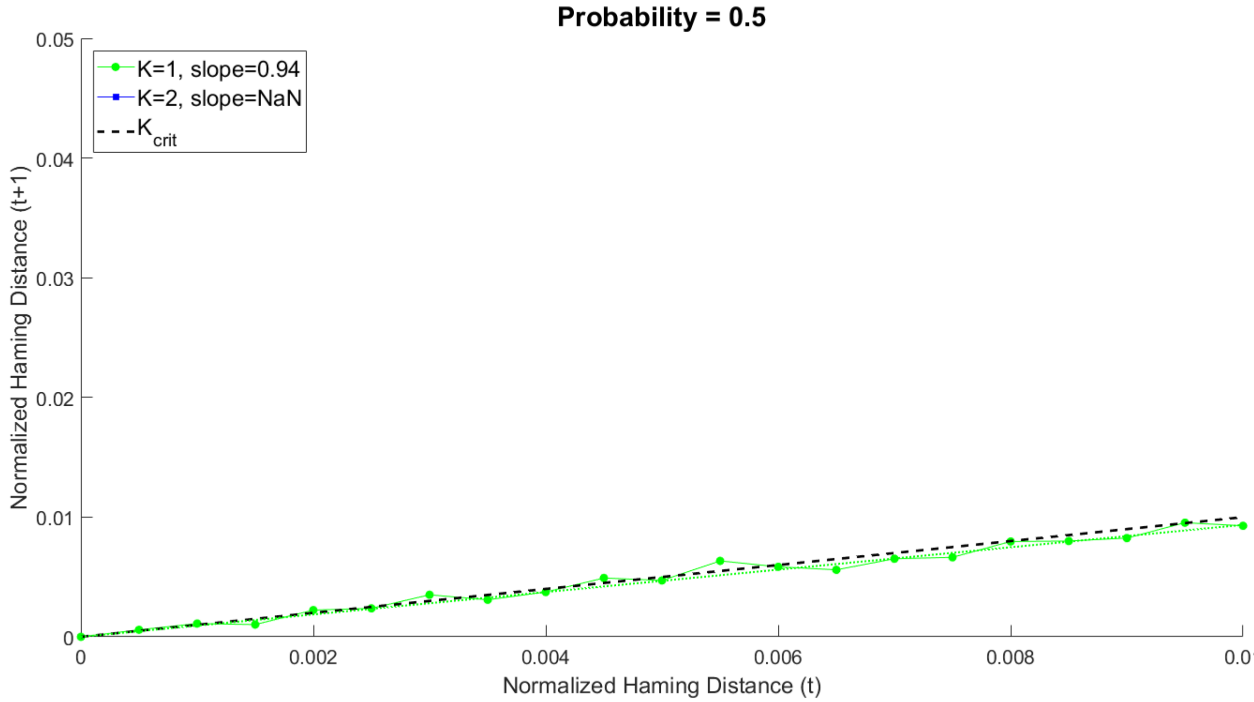
The results shown are a little confusing because they are very close to the ones obtained in the previous assignment. Thus, making any assertion about the dynamical regime of the systems is not justified. This expands to the RBN with power laws.



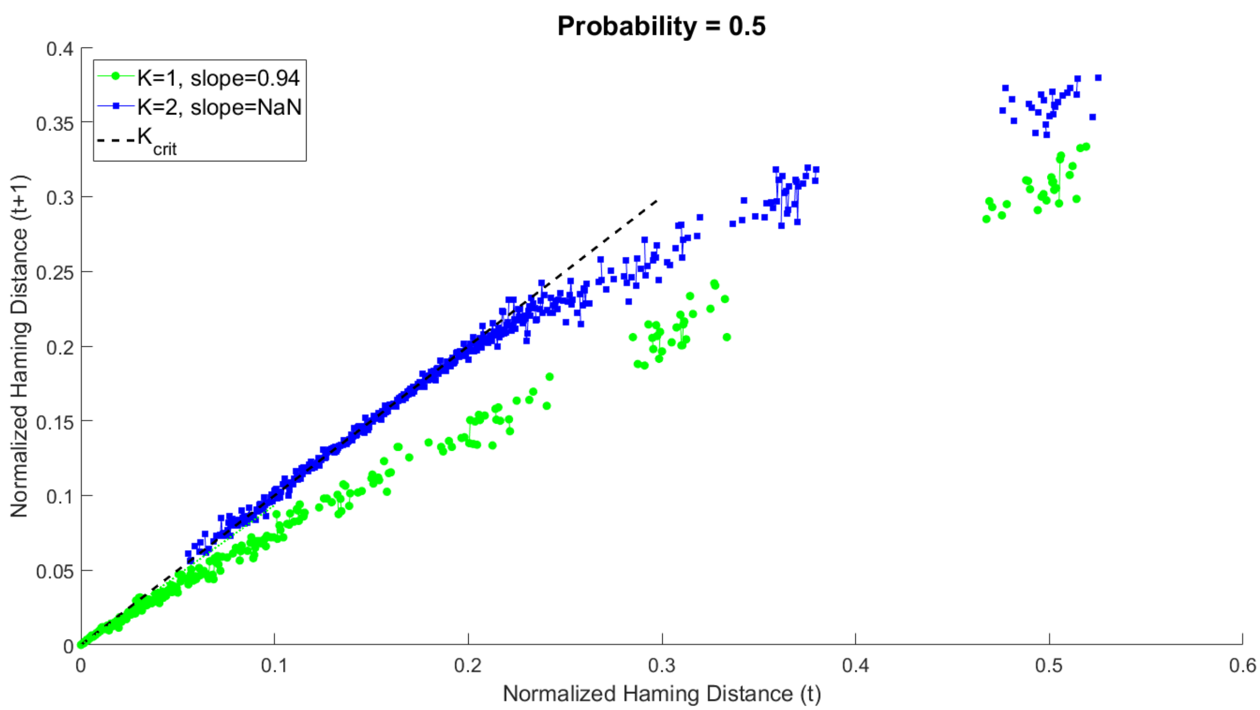
Slope and detail of the Derrida plot close to the origin $p = 0.2$



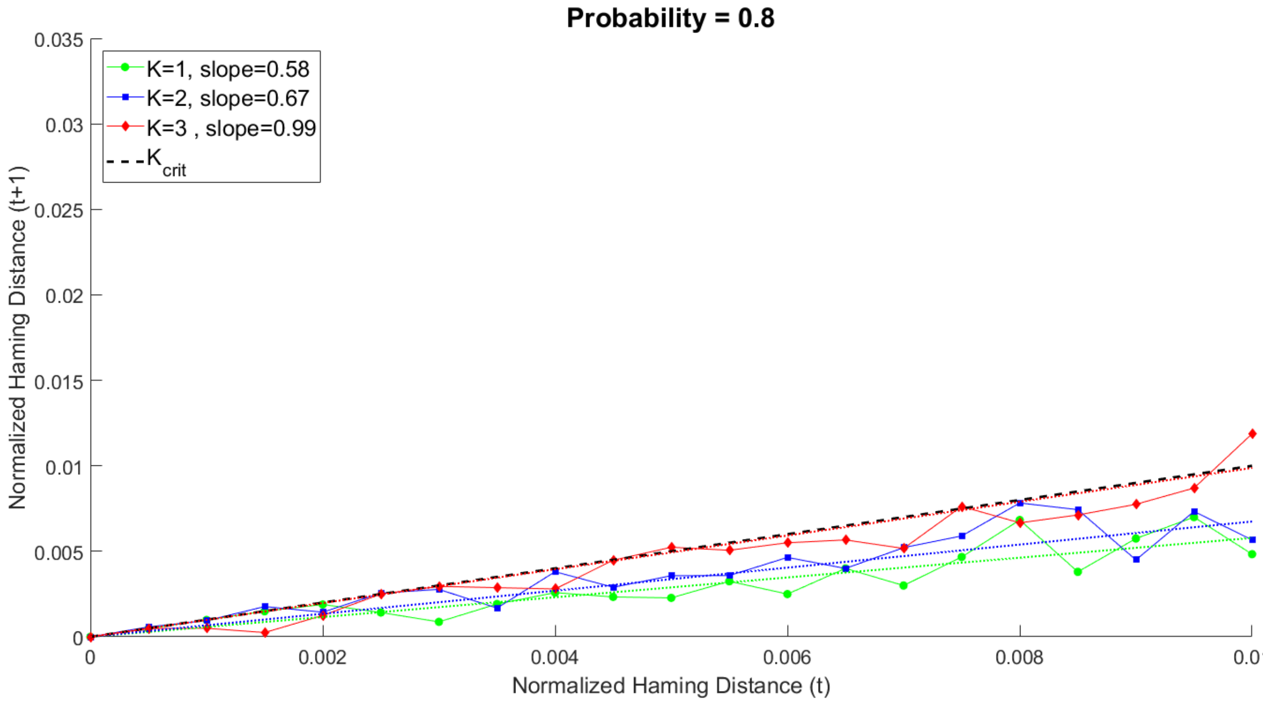
Derrida plot with a broader view of the values $p = 0.2$



Slope and detail of the Derrida plot close to the origin $p = 0.5$



Derrida plot with a broader view of the values $p = 0.5$



Slope and detail of the Derrida plot close to the origin $p = 0.8$

Code listing

`makeRBN.m` file

```matlab
function [inputs,rules]=makeRBN(N,K,p)
% makes an N-K RBN (i.e., assumes K is constant for each of N nodes)
% function [inputs,rules]=makeRBN(N,K,p);
%
% INPUT PARAMETERS
% N: number of nodes
% K: average indegree
% p: probability of 1's in the rules
%
% OUTPUT PARAMETERS:
% inputs: cell aray with N elements, each one with the correspondant
%        connections to other individuals in the network
% rules: cell array for each individual, with a 2^length(real<K>) for each
%        individual in the network, storing a boolean variable

% AUTHOR: Maggie Eppstein
% MODIFIED BY: Javier Lobato 05/02/2018 (adaptation to use Poisson indegree
%        distributed Random Boolean Networks)

% Force this function to clear and re-initialize its persistent variables
clear evolveRBN

% If the probability is not specified, set it as 0.5
if nargin < 3
    p=0.5;
end

% INDIVIDUALS CREATION
% Preallocation of the cell array
inputs = cell(N,1);
% Loop over the whole cell array
for node = 1:N
    % Get the number of connections from a Poisson random distribution
    nodalK = poissrnd(K);
    % Avoid a random number zero and connections bigger than the number of
    % individuals (remove possibility of double connection from one
    % individual to another in the same direction - in opposite directions
    % they are allowed)
    while nodalK > N || nodalK == 0
        % If the number was constrained, get another value
        nodalK = poissrnd(K);
    end
    % Get a random permutation of nodalK individuals of the N possible
    % individuals in the network. The matrix is sorted to have the
    % connections ordered from low to high. The result is then transposed
    % This guarantees that all inputs are unique
    inputs{node}=sort(randperm(N,nodalK))';
end

% RULE SET CREATION
% Preallocation of the rules cell-array
```

```
rules = cell(N,1);
% Loop over all individuals
for node=1:N
    % And assign a random rule depending on the number of conections that
    % cetain individual has by getting a random sequence of numbers of
    % dimension (2^length, 1) and returning a boolean array by comparing
    % the results with a given probabilty
    rules{node}=rand(2.^length(inputs{node}),1)<p;
end
```

`evolveRBN.m` file

```
function [statematrix]=evolveRBN(inputs,rules,startstates,maxit)
% evolveRBN: evolves a RBN certain number of maxit timesteps
% function [statematrix]=evolveRBN(inputs,rules,startstates,maxit)
%
% INPUT PARAMETERS
% inputs: K rows X N column matrix of integers in range [1..N]
%            (each column holds indeces of which nodes point to that node)
% rules: cell array for each individual, with a 2^length(real<K>) for each
%        individual in the network, storing a boolean variable
% startstates: 1 X N binary vector of initial condition
% maxit: number of iterations (optional: default is 1)
%
% OUTPUT PARAMETER:
% statematrix: maxit X N with states along timesteps

% Author: Maggie Eppstein
% MODIFIED BY: Javier Lobato 05/02/2018 (adaptation to use Poisson indegree
%        distributed Random Boolean Networks)

% Set number of iteratiosn to 1 if it is not specified
if nargin < 4
    maxit = 1;
end

% Preallocate the output matrix for increase efficiency
statematrix=zeros(maxit+1,size(startstates,2));

% Assign the initial states to the state matrix in the first position
statematrix(1,:)=startstates;

% Loop over desired timesteps
for t=1:maxit
    % Loop over every element of the network
    for n=1:length(inputs)
        % Compute the index of the rule that must be applied depending on
        % the value of the correspondant index in the previous timestep
        index = bi2de(fliplr(statematrix(t,inputs{n})))+1;
        % Assign the new state into the state matrix
        statematrix(t+1,n)=rules{n}(index);
    end
end
```

newRBN.m file

```
function [inputs,rules,statematrix]=newRBNrun(N,K,maxit,p)
% RBN driver: creates a new NK-RBN with specified p, runs it for maxit its from a random I.C., and
↪  displays results
% [inputs,rules,statematrix]=newRBNrun(N,K,maxit,p)
%
% INPUT PARAMETERS
% N: number of nodes, it must be larger than K
% K: constant in-degree for each node
% maxit: number of timesteps to run (1 if not specified)
% p: probability of ones in each output function (optional: default = 0.5)
%
% OUTPUT PARAMETERS:
% inputs: cell array with N elements, each one with a length given by a
%       Poisson distribution (with value k)
% rules: cell array for each individual, with a 2^length(real<K>) for each
%       individual in the network, storing a boolean variable
% statematrix: maxit X N with the states of the RBN in time

% AUTHOR: Maggie Eppstein

if nargin < 4
    p=0.5; %default prob of 1's
end

startstate=rand(1,N)<0.5; % Create a random initial condition
[inputs,rules]=makeRBN(N,K,p); % Create new NK-RBN
statematrix=evolveRBN(inputs,rules,startstate,maxit); % Evolve the RBN
```

runRBN.m file

```
function [meanData]=runRBN(N, K, p, time, rep)
%RUNRBN Run a Random Boolean Network certain number of times
% INPUTS:
% N        = size of the Random Boolean Network
% K        = average in-degree of the network
% p        = probability of 1's activation in the output
% time     = maximum simulation time
% rep      = number of times that the NK-RBN simulation will be done
% OUTPUTS:
% meanData = mean value for each Hamming distance
% Javier Lobato & Alberto Vidal, created on 2018/04/22

% In order to compute the Hamming distance in t+1 for each Hamming distance
% in t, an average between different runs must be done. Different timesteps
% may have the same Hamming distances in t but different Hamming distances
% in t+1. To do the average of all Hamming distances in t+1 for all the
% timesteps and all the runs, the next code creates two vectors (HD and nHD)
% with N+1 elements. Given that the size N determines the maximum increment
% of Hamming distances, each increment will have an element preallocated in
% the vector. In other words, if N=10, the possible normalized Hamming
% distances will be [0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0], having
% for each one an element in HD and another respective element in nHD. The
% code works the following way:
%   - Run a simulation of the NK-RBN, saving the state matrix
%   - For each element of the of the state matrix, compute the Hamming
%     distance (HD for time t)
%   - Refer to the element corresponding to HD_t in the matrix HD and nHD
%   - Store in that element of HD the value of the Hamming distance for t+1
%   - Also add 1 to the corresponding element in nHD
%   Loop over the previous items for each repetition, computing the mean of
%   all values in HD
    % Preallocation of matrix
    HD = zeros([N+1,1]);
    nHD = zeros([N+1,1]);
    % Let's loop over the number of repetitions
    for j=1:rep
        % Run the NK-RBN simulation
        [~, ~, sm] = newRBNrun(N,K,time,p);
        % Loop over each element of the state-matrix
        for i=1:time-2
            % Compute the Hamming distance
            hdt = (sum(abs(sm(i+1,:)-sm(i,:))))/(N);
            % Get the correspondant index for the Hamming distance in t
            index = round(hdt*N+1);
            % Add to HD(index) the Hamming distance in t+1
            HD(index) = HD(index) + (sum(abs(sm(i+2,:)-sm(i+1,:))))/(N);
            % Add to nHD(index) 1 to count the occurences of HD_t
            nHD(index) = nHD(index) + 1;
        end
    end
    % Once the loop is over, compute and return the mean
    meanData = HD ./ nHD;
end
```

HW5b_JavierLobato.m file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              HOMEWORK #5.B                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Javier Lobato, created on 2018/05/01

% Let's clear the environment
clear all; clc; close all;

%% Section 2
% Let's define first the cell array with the individuals
C = cell(3,1);
% Each element i in the cell array contains the individuals that the
% individual i is point towards
C{1} = [2,3];
C{2} = [1,2,3];
C{3} = [1];

% Let's hardcode also the rules for this second section as given for HW5a
rules = cell(3,1);
rules{1} = [0,0,1,1];
rules{2} = [0,0,1,0,1,1,0,1];
rules{3} = [0,1];

% Initial vectors are given directly to the function. Testing one by one
% all possible inputs to see if the model is correctly working, the state
% matrix will be given as an output
B = zeros([8,2,3]);
B(1,:,:) = evolveRBN(C,rules,[0,0,0]);
B(2,:,:) = evolveRBN(C,rules,[0,0,1]);
B(3,:,:) = evolveRBN(C,rules,[0,1,0]);
B(4,:,:) = evolveRBN(C,rules,[0,1,1]);
B(5,:,:) = evolveRBN(C,rules,[1,0,0]);
B(6,:,:) = evolveRBN(C,rules,[1,0,1]);
B(7,:,:) = evolveRBN(C,rules,[1,1,0]);
B(8,:,:) = evolveRBN(C,rules,[1,1,1]);
% The state matrix for the system (as proposed in the previous assignment)
% is obtained
outputMat = reshape(B(:,2,:),[8,3])


%% SECTION 3 - Computing
% Select the number of times that the case will be run
runs = 100;
% Select the representative number of individuals
representativeN = [2,5,8,10,20,30,40,50,60,70,80,90,100,125,150,175,200,225,250];
% Select different k values
Kset = [2,5,8];
% Preallocate space to save the values obtained in each simulation
section3 = zeros([length(Kset),length(representativeN),length(runs)]);

% Loop over all values of k
for k=1:length(Kset)
    % Loop over all possible number of individuals
    for N=1:length(representativeN)
```

```
        % Repeat the same N-K simulation certain number of times
        for run=1:runs
            % Call the funciton to create a set of N individuals with a
            % target K. Probability value doesn't matter because it only
            % affects the creation of the rules
            ind = makeRBN(representativeN(N),Kset(k),0);
            % Create a variable to store the numbe of connections of each
            % cell (length of each element in the ind{cell array})
            realK = 0;
            % length(ind) = N but for code clarity it is kept the other way
            for i=1:length(ind)
                realK = realK+length(ind{i});
            end
            % Assign the value to the correspondant element in the array
            section3(k,N,run) = realK/length(ind);
        end
    end
end


%% SECTION 3 - Plotting
% Compute the mean and standard deviation of the data obatined before,
% averaging for all runs
meanS3 = mean(section3,3);
stdS3 = std(section3,[],3);

% Plot the results for all three targeted values of k
subplot(1,3,1)
hold on
errorbar(representativeN,meanS3(1,:),stdS3(1,:),'k','LineWidth',1)
% Line that represents the target k
plot([0,max(representativeN)],[2,2],'r--','LineWidth',2)
hold off
title(['Target <k>=', num2str(Kset(1))])
xlabel('N')
ylabel('<K>')
set(gca, 'FontSize',18)
xlim([0,max(representativeN)])

subplot(1,3,2)
hold on
errorbar(representativeN,meanS3(2,:),stdS3(2,:),'k','LineWidth',1)
% Line that represents the target k
plot([0,max(representativeN)],[5,5],'r--','LineWidth',2)
hold off
title(['Target <k>=', num2str(Kset(2))])
xlabel('N')
ylabel('<K>')
set(gca, 'FontSize',18)
xlim([0,max(representativeN)])

subplot(1,3,3)
hold on
errorbar(representativeN,meanS3(3,:),stdS3(3,:),'k','LineWidth',1)
% Line that represents the target k
plot([0,max(representativeN)],[8,8],'r--','LineWidth',2)
hold off
```

```
title(['Target <k>=', num2str(Kset(3))])
xlabel('N')
ylabel('<K>')
set(gca, 'FontSize',18)
xlim([0,max(representativeN)])


%% SECTION 4 - K variation
% Appropiate N value (although higher values give better results)
N = 150;
% Different target K
K1 = 2;
K2 = 5;
K3 = 8;

% Create a representative RBN with desired values
[ind1,~] = makeRBN(N,K1,p1);
[ind2,~] = makeRBN(N,K2,p1);
[ind3,~] = makeRBN(N,K3,p1);

% Preallocate space for data analysis of each RBN
occurencesK1 = zeros([length(ind1),1]);
occurencesK2 = zeros([length(ind2),1]);
occurencesK3 = zeros([length(ind3),1]);

% Loop the individuals matrix to get the length of each individual
for i=1:length(ind1)
    occurencesK1(i) = length(ind1{i});
    occurencesK2(i) = length(ind2{i});
    occurencesK3(i) = length(ind3{i});
end

% Get the correct value of the Poisson distribution for each K
y1 = poisspdf(sort(unique(occurencesK1)),K1);
y2 = poisspdf(sort(unique(occurencesK2)),K2);
y3 = poisspdf(sort(unique(occurencesK3)),K3);

% Plotting the results for each K
subplot(1,3,1)
% The results are normalized to sum 1 (and compare them with true Poisson)
histogram(occurencesK1,sort(unique(occurencesK1))-0.5,...
    'Normalization','probability')
hold on
% Including the correct Poisson distribution
plot(sort(unique(occurencesK1)),y1,'o-','Color',[0.3,0.3,0.3],'Linewidth',1)
hold off
title(['Target <k>=', num2str(K1), '  N=', num2str(N),...
    '  Mean=', num2str(mean(occurencesK1),3)])
xlabel('<K>')
ylabel('Frequency')
set(gca, 'FontSize',16)
subplot(1,3,2)
% The results are normalized to sum 1 (and compare them with true Poisson)
histogram(occurencesK2,sort(unique(occurencesK2))-0.5,...
    'Normalization','probability')
hold on
% Including the correct Poisson distribution
```

```
plot(sort(unique(occurencesK2)),y2,'o-','Color',[0.3,0.3,0.3],'Linewidth',1)
hold off
title(['Target <k>=', num2str(K2), '  N=', num2str(N),...
    '  Mean=', num2str(mean(occurencesK2),3)])
xlabel('<K>')
ylabel('Frequency')
set(gca, 'FontSize',16)
subplot(1,3,3)
% The results are normalized to sum 1 (and compare them with true Poisson)
histogram(occurencesK3,sort(unique(occurencesK3))-0.5,...
    'Normalization','probability')
hold on
% Including the correct Poisson distribution
plot(sort(unique(occurencesK3)),y3,'o-','Color',[0.3,0.3,0.3],'Linewidth',1)
hold off
title(['Target <k>=', num2str(K3), '  N=', num2str(N),...
    '  Mean=', num2str(mean(occurencesK3),3)])
xlabel('<K>')
ylabel('Frequency')
set(gca, 'FontSize',16)


%% SECTION 4 - p variation
% Appropiate N value
N = 150;
% Different values of probability to be tested
P1 = 0.2;
P2 = 0.5;
P3 = 0.8;

% Get the rules to each probability with different K
[~,rule1] = makeRBN(N,K2,P1);
[~,rule2] = makeRBN(N,K2,P2);
[~,rule3] = makeRBN(N,K2,P3);
[~,rule4] = makeRBN(N,K3,P1);
[~,rule5] = makeRBN(N,K3,P2);
[~,rule6] = makeRBN(N,K3,P3);

% Preallocation of space for each stored sets of rules
occurencesP1 = zeros([length(rule1),1]);
occurencesP2 = zeros([length(rule2),1]);
occurencesP3 = zeros([length(rule3),1]);
occurencesP4 = zeros([length(rule4),1]);
occurencesP5 = zeros([length(rule5),1]);
occurencesP6 = zeros([length(rule6),1]);

% Get the number of 1's in for each individual normalized with the length
% of each individual
for i=1:length(rule1)
    occurencesP1(i) = sum(rule1{i})/length(rule1{i});
    occurencesP2(i) = sum(rule2{i})/length(rule2{i});
    occurencesP3(i) = sum(rule3{i})/length(rule3{i});
    occurencesP4(i) = sum(rule4{i})/length(rule4{i});
    occurencesP5(i) = sum(rule5{i})/length(rule5{i});
    occurencesP6(i) = sum(rule6{i})/length(rule6{i});
end

```

```
% Plotting the results as histograms
subplot(2,3,1)
% Histograms here are also normalized
histogram(occurencesP1,20,'Normalization','probability')
title(['P=', num2str(P1), '  K=', num2str(K2), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
set(gca, 'FontSize',16)
xlim([0,1])
subplot(2,3,2)
% Histogram is normalized
histogram(occurencesP2,20,'Normalization','probability')
title(['P=', num2str(P2), '  K=', num2str(K2), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
set(gca, 'FontSize',16)
xlim([0,1])
subplot(2,3,3)
% Histogram is normalized
histogram(occurencesP3,20,'Normalization','probability')
title(['P=', num2str(P3), '  K=', num2str(K2), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
xlim([0,1])
set(gca, 'FontSize',16)
subplot(2,3,4)
% Histogram is normalized
histogram(occurencesP4,20,'Normalization','probability')
title(['P=', num2str(P1), '  K=', num2str(K3), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
set(gca, 'FontSize',16)
xlim([0,1])
subplot(2,3,5)
% Histogram is normalized
histogram(occurencesP5,20,'Normalization','probability')
title(['P=', num2str(P2), '  K=', num2str(K3), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
set(gca, 'FontSize',16)
xlim([0,1])
subplot(2,3,6)
% Histogram is normalized
histogram(occurencesP6,20,'Normalization','probability')
title(['P=', num2str(P3), '  K=', num2str(K3), '  N=', num2str(N)])
xlabel("Percentage of 1's in the rules")
ylabel('Frequency')
xlim([0,1])
set(gca, 'FontSize',16)


%% Probability of 0.2

% Number of runs
runNo = 30;
% Number of elements in the RBN
N = 2000;
```

```
% Probability
p = 0.2;
% Average indegree number of the nodes
K = [1, 2, 3];

% Calling to the function
p02k1 = runRBN(N, K(1), p, 100, runNo);
p02k2 = runRBN(N, K(2), p, 100, runNo);
p02k3 = runRBN(N, K(3), p, 100, runNo);

%% Plotting of the probability of 0.2

% Computation of the slopes of the lines
pf1 = polyfit(linspace(0,0.01,0.01*N+1)', p02k1(1:0.01*N+1),1);
pf2 = polyfit(linspace(0,0.01,0.01*N+1)', p02k2(1:0.01*N+1),1);
pf3 = polyfit(linspace(0,0.01,0.01*N+1)', p02k3(1:0.01*N+1),1);

% Figure declaration and plotting
figure(1)
hold on
plot(linspace(0,1,N+1), p02k1, 'go-', 'MarkerFaceColor', 'g')
plot(linspace(0,1,N+1), p02k2, 'bs-', 'MarkerFaceColor', 'b')
plot(linspace(0,1,N+1), p02k3, 'rd-', 'MarkerFaceColor', 'r')
plot([0,0.01],[0,0.01],'k--', 'Linewidth', 2)
plot([0,0.1],[0,0.1*pf1(1)] ,'g:', 'Linewidth',1.5)
plot([0,0.1],[0,0.1*pf2(1)] ,'b:', 'Linewidth',1.5)
plot([0,0.1],[0,0.1*pf3(1)] ,'r:', 'Linewidth',1.5)
hold off

% Forcing ticks to certain position
xticks([0.0, 0.002, 0.004, 0.006, 0.008, 0.01])
yticks([0.0, 0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035])
xlim([0,0.01])
ylim([0,0.035])
set(gca, 'FontSize', 16)

% Legend
len = legend(['K=1, slope=', num2str(pf1(1),2)], ...
             ['K=2, slope=', num2str(pf2(1),2)], ...
             ['K=3 , slope=', num2str(pf3(1),2)], ...
              'K_{crit}', 'Location', 'northwest');
set(len, 'FontSize',18)

% Labeling of axis and title
xlabel('Normalized Haming Distance (t)', 'FontSize',18)
ylabel('Normalized Haming Distance (t+1)', 'FontSize',18)
titletex = ['Probability = ', num2str(p)];
title(titletex, 'Fontsize', 22);

%% Probability of 0.5

% Number of elements in the RBN
N = 2000;
% Probability
p = 0.5;
% Average indegree number of the nodes
K = [1, 2];
```

```

% Calling to the function
p05k1 = runRBN(N, K(1), p, 100, runNo);
p05k2 = runRBN(N, K(2), p, 100, runNo);

%% Plotting of the probability of 0.5

% Computation of the slopes of the lines
pf1 = polyfit(linspace(0,0.01,0.01*N+1)', p05k1(1:0.01*N+1),1);
pf2 = polyfit(linspace(0,0.01,0.01*N+1)', p05k2(1:0.01*N+1),1);

% Figure declaration and plotting
figure(2)
hold on
plot(linspace(0,1,N+1), p05k1, 'go-', 'MarkerFaceColor', 'g')
plot(linspace(0,1,N+1), p05k2, 'bs-', 'MarkerFaceColor', 'b')
plot([0,0.01],[0,0.01],'k--', 'Linewidth', 2)
plot([0,0.1],[0,0.1*pf1(1)] ,'g:', 'Linewidth',1.5)
plot([0,0.1],[0,0.1*pf2(1)] ,'b:', 'Linewidth',1.5)
hold off

% Forcing ticks to certain position
xticks([0.0, 0.002, 0.004, 0.006, 0.008, 0.01])
yticks([0.0, 0.01, 0.02, 0.03, 0.04, 0.05])
xlim([0,0.01])
ylim([0,0.05])
set(gca, 'FontSize', 16)

% Legend
len = legend(['K=1, slope=', num2str(pf1(1),2)], ...
             ['K=2, slope=', num2str(pf2(1),2)], ...
              'K_{crit}', 'Location', 'northwest');
set(len, 'FontSize',18)

% Labeling of axis and title
xlabel('Normalized Haming Distance (t)', 'FontSize',18)
ylabel('Normalized Haming Distance (t+1)', 'FontSize',18)
titletex = ['Probability = ', num2str(p)];
title(titletex, 'Fontsize', 22);

%% Probability of 0.8

% Number of elements in the RBN
N = 2000;
% Probability
p = 0.8;
% Average indegree number of the nodes
K = [1, 2, 3];

% Calling to the function
p08k1 = runRBN(N, K(1), p, 100, runNo);
p08k2 = runRBN(N, K(2), p, 100, runNo);
p08k3 = runRBN(N, K(3), p, 100, runNo);

%% Plotting of the probability of 0.8

% Computation of the slopes of the lines
```

```
pf1 = polyfit(linspace(0,0.01,0.01*N+1)', p08k1(1:0.01*N+1),1);
pf2 = polyfit(linspace(0,0.01,0.01*N+1)', p08k2(1:0.01*N+1),1);
pf3 = polyfit(linspace(0,0.01,0.01*N+1)', p08k3(1:0.01*N+1),1);

% Figure declaration and plotting
figure(3)
hold on
plot(linspace(0,1,N+1), p08k1, 'go-', 'MarkerFaceColor', 'g')
plot(linspace(0,1,N+1), p08k2, 'bs-', 'MarkerFaceColor', 'b')
plot(linspace(0,1,N+1), p08k3, 'rd-', 'MarkerFaceColor', 'r')
plot([0,0.01],[0,0.01],'k--', 'Linewidth', 2)
plot([0,0.1],[0,0.1*pf1(1)] ,'g:', 'Linewidth',1.5)
plot([0,0.1],[0,0.1*pf2(1)] ,'b:', 'Linewidth',1.5)
plot([0,0.1],[0,0.1*pf3(1)] ,'r:', 'Linewidth',1.5)
hold off

% Forcing ticks to certain position
xticks([0.0, 0.002, 0.004, 0.006, 0.008, 0.01])
yticks([0.0, 0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035])
xlim([0,0.01])
ylim([0,0.035])
set(gca, 'FontSize', 16)

% Legend
len = legend(['K=1, slope=', num2str(pf1(1),2)], ...
             ['K=2, slope=', num2str(pf2(1),2)], ...
             ['K=3 , slope=', num2str(pf3(1),2)], ...
              'K_{crit}', 'Location', 'northwest');
set(len, 'FontSize',18)

% Labeling of axis and title
xlabel('Normalized Haming Distance (t)', 'FontSize',18)
ylabel('Normalized Haming Distance (t+1)', 'FontSize',18)
titletex = ['Probability = ', num2str(p)];
title(titletex, 'Fontsize', 22);
```