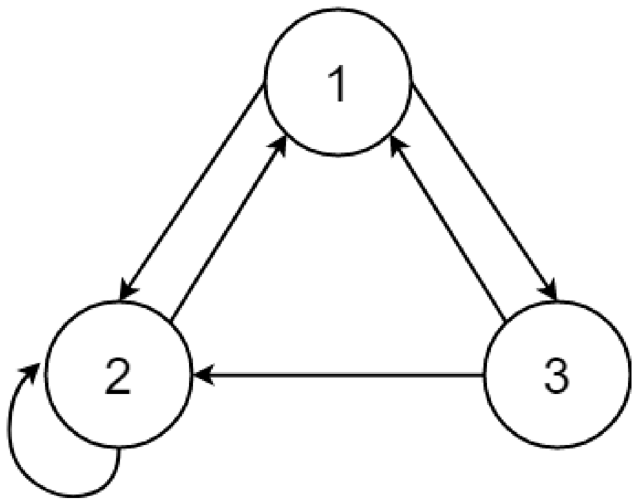
Homework 5.(a)

Modeling Complex Systems, Javier Lobato & Alberto Vidal

Due date: Tuesday, April 24, 2018

Part 1

The Random Boolean Network determined by the given tables is the next one:



The state-space can be summarize in the next table:

T			**T+1**		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

There are three different attractors which are identified as:

- Fixed point: `0 0 1` $\longrightarrow$ `0 0 0` ⟲
- Fixed point: `1 1 1` ⟲
- Periodic attractor of cycle 2: `0 1 0` $\longrightarrow$ `1 1 0` $\longrightarrow$ `1 0 1` $\longrightarrow$ `0 1 1` $\rightleftharpoons$ `1 0 0`

The length and the size of the basin of each attractor are summarized in the next table:

Attractor	Fixed point 1 (000)	Fixed point 2 (111)	Periodic attractor
Length of the basin	1	0	3
Size of the basin	2	1	5

The basin-entropy for this network can be calculated as follows:

$$h(B) = -\sum_{\rho} w_\rho \log w_\rho$$

where w_ρ is the weight if the attractor ρ. One can compute the weight of each attractor as:

$$w_\rho = \frac{l_\rho + n_{bas,states}}{2^n}$$

$$w_{\rho_1} = \frac{1+1}{2^3} = 0.25$$

$$w_{\rho_2} = \frac{0+1}{2^3} = 0.125$$

$$w_{\rho_3} = \frac{2+3}{2^3} = 0.625$$

$$\sum_{\rho} w_\rho = w_{\rho_1} + w_{\rho_2} + w_{\rho_3} = 1$$

Plugging this values in the basin-entropy equation yields:

$$h(B) = -\left[\frac{1}{4}\log\left(\frac{1}{4}\right) + \frac{1}{8}\log\left(\frac{1}{8}\right) + \frac{5}{8}\log\left(\frac{5}{8}\right)\right] = 0.9$$

Rather than trying a set of random values attempting to obtain the maximum and minimum of the function, one can think that the maximum h value will take place where there are as many attractors as states. On the contrary, the minimum h value will be obtained when there is only one attractor. Therefore, the maximum and minimum value for 2^3 states are:

$$h(B)_{min} = -\sum_{\rho=1}^{8} w_\rho \log w_\rho = 1\left[1\log\left(1\right)\right] = 0$$

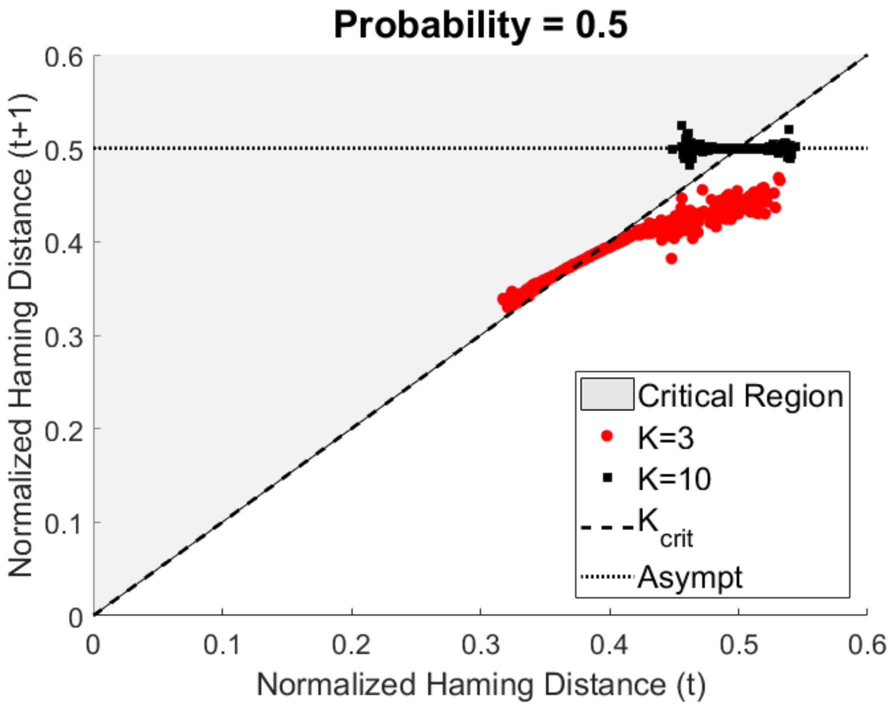
$$h(B)_{max} = -\sum_{\rho=1}^{8} w_\rho \log w_\rho = 8\left[\frac{1}{8}\log\left(\frac{1}{8}\right)\right] = 2.079$$

Comparing the value of our network with the upper and lower limits, the system is located in the middle part of the range. This means that our system is neither completely deterministic nor totally random. There are three possibilities for a given initial state, although the probability of ending up in one of the attractors is much greater than the rest.

Part 2

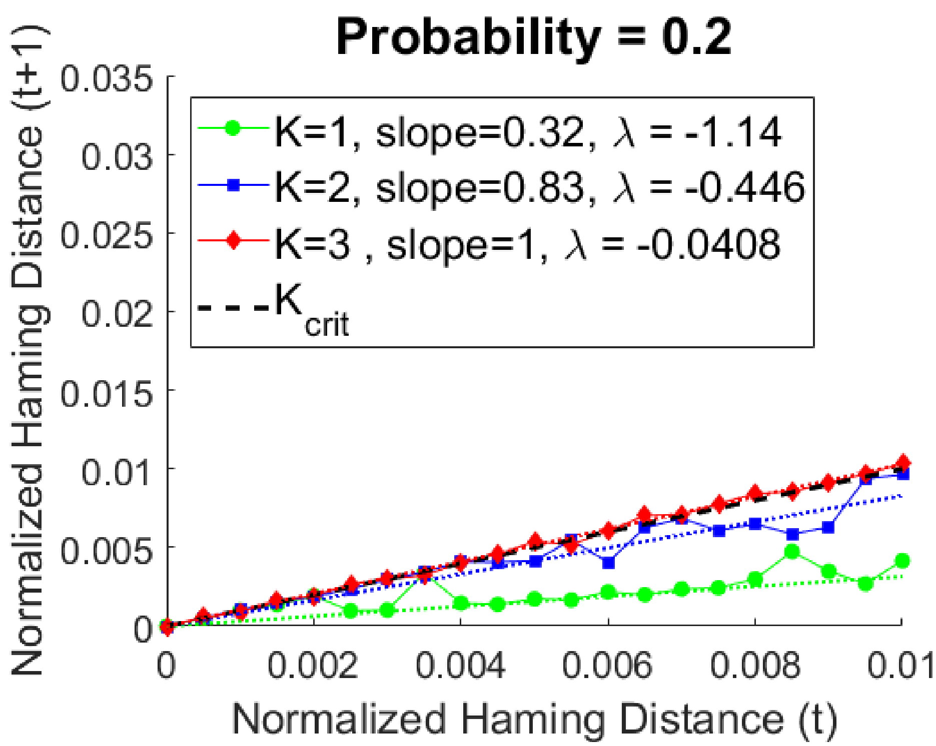
Prior to showing the plots, it must be stated that the figures have been constructed using 200 repetitions since using 25 repetitions did not yield as many points as the reference figure includes.

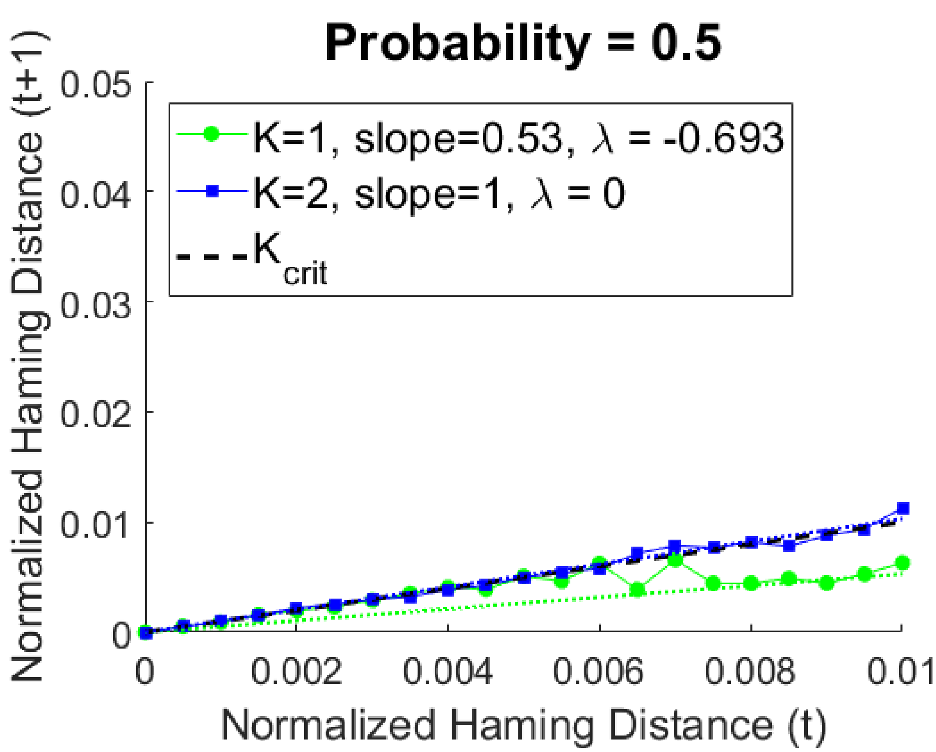
Another aspect which needs to be explained is the fact that the plots do not contain so many curves. This is due to fact that we were not able to obtain any point close to the origin if the value of k exceeded the criticality. As a matter of fact, the points obtained with k above k_{critic} tend to gather around a value of the Normalized Hamming Distance of 0.5. This can be shown in the next figure:

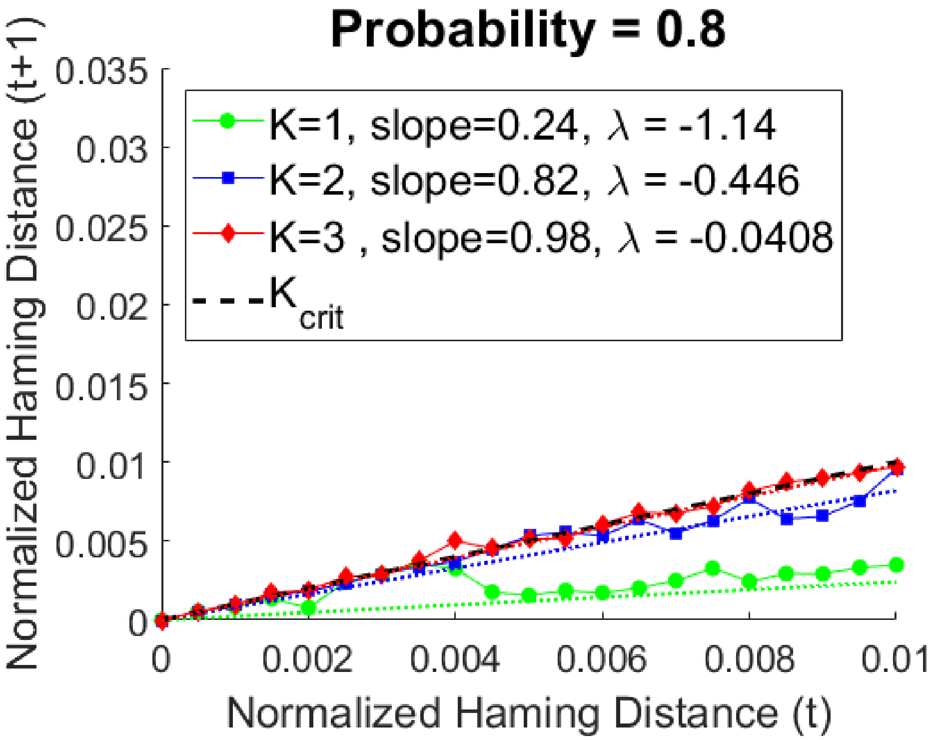


Note that $p = 0.2$ and $p = 0.8$ yield the same value of criticality, $k_{critic} = 3.125$, since this is a Boolean network with random topology. Therefore, the curve constructed using $k = 10$ will not appear in any of those two plots. Similarly, for the case of probability of 0.5, the value of the critical average in-degree of the nodes is 2. Therefore, the curves related to $k = 3$ and $k = 10$ will not appear.

With this being said, the figures obtained for the different values of probability are:







As it can be seen, the slope lines (dotted lines in the figures) pass through the origin. When the line fitting was calculated, the y-intersect coefficient, b, was of the order of 10^{-4}. Consequently, the line was constructed as $y = a \cdot x$, neglecting the value of b for code simplicity.

The value of the probability will determine the dynamic regime of the system since the criticality is a function of the probability. As it was said before, $p = 2$ and $p = 8$ yield the same k_{critic}. Therefore, the dynamical behavior of the systems for these probabilities is the same. As the criticality rises, the slope of the line with a given value of k will go down. This means that this regime is further from the critical behavior, which corresponds to slope 1.

In terms of the code, the functions provided have not been modified, although the `plotStates.m` has been commented to avoid creating a figure each time an RBN was run. A new function has been created to compute the Random Boolean Network N times and average the values of the Normalized Hamming Distance. The driver includes the code to plot the results.

Code listing

This section contains the different listings of code used throughout the homework. Although all the functions have been included they have not been modified. The only new code is the driver `HW5a_JavierLobato_AlbertoVidal.m` and the function `runRBN.m`.

`HW5a_JavierLobato_AlbertoVidal.m` file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               HOMEWORK #5.A                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Javier Lobato & Alberto Vidal, created on 2018/04/22

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

% Definition of the number of runs for each set
runNo = 200;

%% Probability of 0.2

% 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, 500, runNo);
p02k2 = runRBN(N, K(2), p, 200, runNo);
p02k3 = runRBN(N, K(3), p, 500, 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);

% Lambda value calculation
lambda = log(2*p*(1-p).*K);

% 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])
set(gca, 'FontSize', 16)

% Legend
len = legend(['K=1, slope=', num2str(pf1(1),2),', \lambda = ', num2str(lambda(1),3)], ...
             ['K=2, slope=', num2str(pf2(1),2),', \lambda = ', num2str(lambda(2),3)], ...
             ['K=3 , slope=', num2str(pf3(1),2),', \lambda = ', num2str(lambda(3),3)], ...
              '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);
xlim([0,0.01])
ylim([0,0.035])

%% 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, 500, runNo);
p05k2 = runRBN(N, K(2), p, 200, 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);

% Lambda value calculation
lambda = log(2*p*(1-p).*K);

% 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),', \lambda = ', num2str(lambda(1),3)], ...
             ['K=2, slope=', num2str(pf2(1),2),', \lambda = ', num2str(lambda(2),3)], ...
              '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, 500, runNo);
p08k2 = runRBN(N, K(2), p, 200, runNo);
p08k3 = runRBN(N, K(3), p, 500, 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);

% Lambda value calculation
lambda = log(2*p*(1-p).*K);

% 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),', \lambda = ', num2str(lambda(1),3)], ...
             ['K=2, slope=', num2str(pf2(1),2),', \lambda = ', num2str(lambda(2),3)], ...
             ['K=3 , slope=', num2str(pf3(1),2),', \lambda = ', num2str(lambda(3),3)], ...
              '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);

%% Above critical values (K=3 and K=10 for p=0.5)

% Number of elements in the RBN
N = 2000;
% Probability
p = 0.8;

% Calling to the function
p05k3 = runRBN(N, 3, p, 500, runNo);
p05k10 = runRBN(N, 10, p, 200, runNo);

%% Plotting of the above critical values

% Figure declaration and plotting
figure(4)
hold on
x = [-0.1,0.7];
X=[x,fliplr(x)];
Y=[[-0.1,0.7],fliplr([0.7,0.7])];
h = fill(X,Y,[0.9 0.9 0.9]);
set(h,'facealpha',.5)
plot(linspace(0,1,N+1), p05k3, 'ro', 'MarkerFaceColor', 'r')
plot(linspace(0,1,N+1), p05k10, 'ks', 'MarkerFaceColor', 'k')
plot([0,0.6],[0,0.6],'k--', 'Linewidth', 2)
plot([0,0.6],[0.5,0.5],'k:', 'Linewidth', 2)
hold off

% Forcing figure limits
xlim([0,0.6])
ylim([0,0.6])
set(gca, 'FontSize', 16)

% Legend
len = legend(['Critical Region'],['K=3'], ['K=10'], ['K_{crit}'], ['Asympt'], ...
              'Location', 'southeast');
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);
```

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 corresponding 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 occurrences of HD_t
            nHD(index) = nHD(index) + 1;
        end
```

```
    end

    % Once the loop is over, compute and return the mean
    meanData = HD ./ nHD;

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
% K: constant in-degree for each node
% maxit: number of timesteps to run
% p: probability of ones in each output function (optional: default = 0.5)
%
% OUTPUT PARAMETERS:
% inputs: K rows by N columns
%           (each column holds indeces of which nodes point to that node)
% rules: 2^K rows by N columns
%           (each column holds the boolean outputs for each of the 2^K
%           possible inputs, in increasing binary order)
% statematrix: maxit X N

% AUTHOR: Maggie Eppstein

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

startstate=rand(1,N)<0.5; %random initial condition
[inputs,rules]=makeRBN(N,K,p); %create new NK-RBN
statematrix=evolveRBN(inputs,rules,startstate,maxit); %run it for maxit timesteps
% plotStates(statematrix); %plot the results
% title(['New RBN: N=',num2str(N),' K=',num2str(K), ' p=',num2str(p)]);
```

makeRBN.m file

```
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: in-degree of each node
% p: probability of 1's in the boolean output function
%
% OUTPUT PARAMETERS:
% inputs: K rows by N columns
%           (each column holds indeces of which nodes point to that node)
% rules: 2^K rows by N columns
%           (each column holds the boolean outputs for each of the 2^K
%           possible inputs, in increasing binary order)

% AUTHOR: Maggie Eppstein

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

if nargin < 3
    p=0.5;
end

% make the random topology (fixed in-degree of K, Poisson distributed out-degree)
inputs=zeros(K,N); % pre-allocate for efficiency
for node=1:N
    inputs(:,node)=randperm(N,K)'; %guarantees all inputs are unique
end
% inputs=randi([1 N],K,N); % this vectorizes the above loop, but doesn't
% guarantee no duplicate inputs, so it shouldn't be used

rules=rand(2.^K,N)<p; %make the random boolean output functions
```

evolveRBN.m file

```
function [statematrix]=evolveRBN(inputs,rules,startstates,maxit)
% evolveRBN: evolves an N-K RBN maxit timesteps; assumes K is constant for each of the N nodes
% 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: 2^K rows by N column binary matrix
%           (each column holds the boolean outputs for each of the 2^K
%           possible inputs, in increasing binary order)
% startstates: 1 X N binary vector of initial condition
% maxit: number of iterations (optional: default is 1)
%
% OUTPUT PARAMETER:
% statematrix: maxit X N
```

```

% Author: Maggie Eppstein

persistent N K inputvals coloffsets %persistent means these persist in memory between function calls

if isempty(N) % don't bother to calculate these persistent things more than once
    [K,N]=size(inputs);
    inputvals=[2.^(K-1:-1:0)]; % decimal values of binary inputs
    coloffsets=2^K.*(0:N-1)+1; % 1-D index into 1st row in every col
end

if nargin < 4
    maxit = 1; %default is to evolve only 1 timestep
end

% pre-allocate the statematrix for efficiency
statematrix=zeros(maxit+1,size(startstates,2));
statematrix(1,:)=startstates;

for t=1:maxit
    states=statematrix(t,:);

    statematrix(t+1,:)=rules(inputvals*states(inputs)+coloffsets);
    % THE ABOVE LINE IMPLEMENTS THE EQUIVALENT OF THE FOLLOWING 4 STEPS;
%     inputstates=states(inputs); % get K input states to each node
%     rowindex=inputvals*inputstates; % compute the decimal equivalent of the binary input string
%     ruleindex=rowindex+coloffsets; % convert col indeces into 1-D matrix indeces
%     newstates=rules(ruleindex); % extract the appropriate output values
end
```